The Ciao System

A New Generation, Multi-Paradigm Programming Language and Environment
(Including a State-of-the-Art ISO-Prolog)
REFERENCE MANUAL
The Ciao Documentation Series
`http://www.ciaohome.org/`
Generated/Printed on: 14 June 2011
Technical Report CLIP 3/97-1.13#1
Version 1.13#1 (2011/3/15, 20:8:35 CEST)

Edited by:
Francisco Bueno
Daniel Cabeza
Manuel Carro
Manuel Hermenegildo
Pedro López
Germán Puebla

The Computational logic, Languages, Implementation, and Parallelism (CLIP) Lab
`http://www.cliplab.org/`
`webmaster@clip.dia.fi.upm.es`
School of CS, T. U. of Madrid (UPM)
IMDEA Software Institute

Copyright © 1997-2011 Francisco Bueno, Daniel Cabeza, Manuel Carro, Manuel Hermenegildo, P. López, and Germán Puebla This document may be freely read, stored, reproduced, disseminated, translated or quoted by any means and on any medium provided the following conditions are met:

1. Every reader or user of this document acknowledges that is aware that no guarantee is given regarding its contents, on any account, and specifically concerning veracity, accuracy and fitness for any purpose.
2. No modification is made other than cosmetic, change of representation format, translation, correction of obvious syntactic errors, or as permitted by the clauses below.
3. Comments and other additions may be inserted, provided they clearly appear as such; translations or fragments must clearly refer to an original complete version, preferably one that is easily accessed whenever possible.
4. Translations, comments and other additions or modifications must be dated and their author(s) must be identifiable (possibly via an alias).
5. This licence is preserved and applies to the whole document with modifications and additions (except for brief quotes), independently of the representation format.
6. Any reference to the "official version", "original version" or "how to obtain original versions" of the document is preserved verbatim. Any copyright notice in the document is preserved verbatim. Also, the title and author(s) of the original document should be clearly mentioned as such.
7. In the case of translations, verbatim sentences mentioned in (6.) are preserved in the language of the original document accompanied by verbatim translations to the language of the traslated document. All translations state clearly that the author is not responsible for the translated work. This license is included, at least in the language in which it is referenced in the original version.
8. Whatever the mode of storage, reproduction or dissemination, anyone able to access a digitized version of this document must be able to make a digitized copy in a format directly usable, and if possible editable, according to accepted, and publicly documented, public standards.
9. Redistributing this document to a third party requires simultaneous redistribution of this licence, without modification, and in particular without any further condition or restriction, expressed or implied, related or not to this redistribution. In particular, in case of inclusion in a database or collection, the owner or the manager of the database or the collection renounces any right related to this inclusion and concerning the possible uses of the document after extraction from the database or the collection, whether alone or in relation with other documents.

Any incompatibility of the above clauses with legal, contractual or judiciary decisions or constraints implies a corresponding limitation of reading, usage, or redistribution rights for this document, verbatim or modified.

Table of Contents

Summary 1

1 Introduction 3
- 1.1 About this manual 3
- 1.2 About the Ciao development system 3
- 1.3 ISO-Prolog compliance versus extensibility 4
- 1.4 About the name of the System 5
- 1.5 Referring to Ciao 5
- 1.6 Syntax terminology and notational conventions 5
  - 1.6.1 Predicates and their components 5
  - 1.6.2 Characters and character strings 6
  - 1.6.3 Predicate specs 6
  - 1.6.4 Modes 6
  - 1.6.5 Properties and types 6
  - 1.6.6 Declarations 6
  - 1.6.7 Operators 7
- 1.7 A tour of the manual 7
  - 1.7.1 PART I - The program development environment 7
  - 1.7.2 PART II - The Ciao basic language (engine) 8
  - 1.7.3 PART III - ISO-Prolog library (iso) 8
  - 1.7.4 PART IV - Classic Prolog library (classic) 8
  - 1.7.5 PART V - Assertions, Properties, Types, Modes, Comments (assertions) 8
  - 1.7.6 PART VI - Ciao library miscellanea 8
  - 1.7.7 PART VII - Ciao extensions 8
  - 1.7.8 PART VIII - Interfaces to other languages and systems 9
  - 1.7.9 PART IX - Abstract data types 9
  - 1.7.10 PART X - Miscellaneous standalone utilities 9
  - 1.7.11 PART XI - Contributed libraries 9
  - 1.7.12 PART XII - Appendices 9
- 1.8 Acknowledgments 9
- 1.9 Version/Change Log 10

2 Getting started on Un*x-like machines 21
- 2.1 Testing your Ciao Un*x installation 21
- 2.2 Un*x user setup 21
- 2.3 Using Ciao from a Un*x command shell 22
  - 2.3.1 Starting/exiting the top-level shell (Un*x) 22
  - 2.3.2 Getting help (Un*x) 22
  - 2.3.3 Compiling and running programs (Un*x) 22
  - 2.3.4 Generating executables (Un*x) 23
  - 2.3.5 Running Ciao scripts (Un*x) 23
  - 2.3.6 The Ciao initialization file (Un*x) 24
  - 2.3.7 Printing manuals (Un*x) 24
- 2.4 An introduction to the Ciao emacs environment (Un*x) 24
- 2.5 Keeping up to date (Un*x) 25

3 Getting started on Windows machines 27

3.1 Testing your Ciao Win32 installation 27
3.2 Using Ciao from the Windows explorer and command shell 27
3.2.1 Starting/exiting the top-level shell (Win32) 27
3.2.2 Getting help (Win32) 28
3.2.3 Compiling and running programs (Win32) 28
3.2.4 Generating executables (Win32) 28
3.2.5 Running Ciao scripts (Win32) 29
3.2.6 The Ciao initialization file (Win32) 29
3.2.7 Printing manuals (Win32) 29
3.3 An introduction to the Ciao emacs environment (Win32) 29
3.4 Keeping up to date (Win32) 30

PART I - The program development environment 31

4 The stand-alone command-line compiler 33

4.1 Introduction to building executables 33
4.2 Paths used by the compiler during compilation 34
4.3 Running executables from the command line 34
4.4 Types of executables generated 35
4.5 Environment variables used by Ciao executables 37
4.6 Intermediate files in the compilation process 37
4.7 Usage (ciaoc) 38
4.8 Known bugs and planned improvements (`ciaoc`) 40

5 The interactive top-level shell 41

5.1 Shell invocation and startup 41
5.2 Shell interaction 41
5.3 Entering recursive (conjunctive) shell levels 42
5.4 Usage and interface (`toplevel_doc`) 43
5.5 Documentation on exports (`toplevel_doc`) 43
use_module/1 (pred) 43
use_module/2 (pred) 44
ensure_loaded/1 (pred) 44
make_exec/2 (pred) 44
include/1 (pred) 44
use_package/1 (pred) 45
consult/1 (pred) 45
compile/1 (pred) 45
./2 (pred) 45
make_po/1 (pred) 45
unload/1 (pred) 45
set_debug_mode/1 (pred) 46
set_nodebug_mode/1 (pred) 46
make_actmod/2 (pred) 46
force_lazy/1 (pred) 46
undo_force_lazy/1 (pred) 46
dynamic_search_path/1 (pred) 46
(multifile)/1 (pred) 47
5.6 Documentation on internals (`toplevel_doc`) 47
sourcenames/1 (prop) 47

6 The interactive debugger 49

6.1 Marking modules and files for debugging in the top-level debugger 49
6.2 The debugging process 50
6.3 Marking modules and files for debugging with the embedded debugger 50
6.4 The procedure box control flow model 52
6.5 Format of debugging messages 53
6.6 Options available during debugging 54
6.7 Calling predicates that are not exported by a module 56
6.8 Acknowledgements (debugger) 56

7 Predicates controlling the interactive debugger 59

7.1 Usage and interface (`debugger`) 59
7.2 Documentation on exports (`debugger`) 59
call_in_module/2 (pred) 59
breakpt/6 (udreexp) 59
debug/0 (udreexp) 59
debug_module/1 (udreexp) 59
debug_module_source/1 (udreexp) 60
debugging/0 (udreexp) 60
debugrtc/0 (udreexp) 60
get_debugger_state/1 (udreexp) 60
get_debugger_state/1 (udreexp) 60
leash/1 (udreexp) 60
list_breakpt/0 (udreexp) 60
maxdepth/1 (udreexp) 60
nobreakall/0 (udreexp) 60
nobreakpt/6 (udreexp) 60
nodebug/0 (udreexp) 60
nodebug_module/1 (udreexp) 60
nodebugrtc/0 (udreexp) 61
nospy/1 (udreexp) 61
nospyall/0 (udreexp) 61
notrace/0 (udreexp) 61
spy/1 (udreexp) 61
trace/0 (udreexp) 61
tracertc/0 (udreexp) 61
7.3 Documentation on multifiles (`debugger`) 61
define_flag/3 (pred) 61
7.4 Known bugs and planned improvements (`debugger`) 61

8 The script interpreter 63

8.1 How it works 63
8.2 Command line arguments in scripts 64

9 Customizing library paths and path aliases 65

9.1 Usage and interface (`libpaths`) 65
9.2 Documentation on exports (`libpaths`) 65
get_alias_path/0 (pred) 65
9.3 Documentation on multifiles (`libpaths`) 65
file_search_path/2 (pred) 65
library_directory/1 (pred) 66

10 Using Ciao inside GNU emacs 67

10.1 Conventions for writing Ciao programs under Emacs 67
10.2 Checking the installation 68
10.3 Functionality and associated key sequences (bindings) 68
10.4 Syntax coloring and syntax-based editing 68
10.5 Getting on-line help 68
10.6 Loading and compiling programs 69
10.7 Commands available in toplevel and preprocessor buffers 70
10.8 Locating errors and checking the syntax of assertions 72
10.9 Commands which help typing in programs 72
10.10 Debugging programs 72
10.11 Preprocessing programs 73
10.12 Version control 74
10.13 Generating program documentation 77
10.14 Setting top level preprocessor and documenter executables ... 78
10.15 Other commands 78
10.16 Traditional Prolog Mode Commands 79
10.17 Coexistence with other Prolog-like interfaces 79
10.18 Getting the Ciao mode version 79
10.19 Using Ciao mode capabilities in standard shells 79
10.20 Customization 80
10.20.1 Ciao general variables 80
10.20.2 CiaoPP variables 81
10.20.3 LPdoc variables 81
10.20.4 Faces used in syntax-based highlighting (coloring) .. 82
10.21 Installation of the Ciao emacs interface 85
10.22 Emacs version compatibility 87
10.23 Acknowledgments (ciao.el) 87

PART II - The Ciao basic language (engine) 89

11 The module system 91

11.1 Usage and interface (`modules`) 91
11.2 Documentation on internals (`modules`) 91
module/3 (decl) 91
module/2 (decl) 92
package/1 (decl) 92
export/1 (decl) 92
use_module/2 (decl) 93
use_module/1 (decl) 93
import/2 (decl) 93
reexport/2 (decl) 93
reexport/1 (decl) 94
(meta_predicate)/1 (decl) 94
modulename/1 (regtype) 94
metaspec/1 (regtype) 94

12 Directives for using code in other files 97

12.1 Usage and interface (`loading_code`) 97
12.2 Documentation on internals (`loading_code`) 97
ensure_loaded/1 (decl) 97
include/1 (decl) 97
use_package/1 (decl) 97

13 Control constructs/predicates 99

13.1 Usage and interface (`basiccontrol`) 99
13.2 Documentation on exports (`basiccontrol`) 99
,/2 (pred) 99
;/2 (pred) 99
-> /2 (pred) 100
!/0 (pred) 100
(\+)/1 (pred) 100
if/3 (pred) 100
true/0 (pred) 101
fail/0 (pred) 101
repeat/0 (pred) 101
false/0 (pred) 102
otherwise/0 (pred) 102
13.3 Known bugs and planned improvements (`basiccontrol`) 102

14 Basic builtin directives 103

14.1 Usage and interface (`builtin_directives`) 103
14.2 Documentation on internals (`builtin_directives`) 103
(multifile)/1 (decl) 103
(discontiguous)/1 (decl) 103
impl_defined/1 (decl) 103
redefining/1 (decl) 104
initialization/1 (decl) 104
on_abort/1 (decl) 104

15 Basic data types and properties 105

15.1 Usage and interface (`basic_props`) 105
15.2 Documentation on exports (`basic_props`) 105
term/1 (regtype) 105
int/1 (regtype) 106
nnegint/1 (regtype) 106
flt/1 (regtype) 107
num/1 (regtype) 107
atm/1 (regtype) 108
struct/1 (regtype) 108
gnd/1 (regtype) 109
gndstr/1 (regtype) 109
constant/1 (regtype) 110
callable/1 (regtype) 110
operator_specifier/1 (regtype) 110
list/1 (regtype) 111
list/2 (regtype) 112
nlist/2 (regtype) 112
member/2 (prop) 113
sequence/2 (regtype) 113
sequence_or_list/2 (regtype) 113
character_code/1 (regtype) 114
string/1 (regtype) 114
num_code/1 (regtype) 115
predname/1 (regtype) 115
atm_or_atm_list/1 (regtype) 115
compat/2 (prop) 116
inst/2 (prop) 116
iso/1 (prop) 116

deprecated/1 (prop) ... 117
not_further_inst/2 (prop) ... 117
sideff/2 (prop) ... 117
(regtype)/1 (prop) ... 118
native/1 (prop) ... 118
native/2 (prop) ... 118
no_rtcheck/1 (prop) ... 118
eval/1 (prop) ... 118
equiv/2 (prop) ... 118
bind_ins/1 (prop) ... 119
error_free/1 (prop) ... 119
memo/1 (prop) ... 119
filter/2 (prop) ... 119
flag_values/1 (regtype) ... 119
pe_type/1 (prop) ... 119
15.3 Known bugs and planned improvements (basic_props) ... 119

16 Extra-logical properties for typing ... 121
16.1 Usage and interface (term_typing) ... 121
16.2 Documentation on exports (term_typing) ... 121
var/1 (prop) ... 121
nonvar/1 (prop) ... 122
atom/1 (prop) ... 122
integer/1 (prop) ... 123
float/1 (prop) ... 124
number/1 (prop) ... 125
atomic/1 (prop) ... 125
ground/1 (prop) ... 126
type/2 (prop) ... 127
16.3 Known bugs and planned improvements (term_typing) ... 127

17 Basic term manipulation ... 129
17.1 Usage and interface (term_basic) ... 129
17.2 Documentation on exports (term_basic) ... 129
= /2 (prop) ... 129
\= /2 (pred) ... 129
arg/3 (pred) ... 130
functor/3 (pred) ... 131
=.. /2 (pred) ... 132
non_empty_list/1 (regtype) ... 132
copy_term/2 (pred) ... 132
copy_term_nat/2 (pred) ... 133
C/3 (pred) ... 133
const_head/1 (prop) ... 133
list_functor/1 (regtype) ... 134
17.3 Known bugs and planned improvements (term_basic) ... 134

18 Comparing terms 135

18.1 Usage and interface (`term_compare`) 135
18.2 Documentation on exports (`term_compare`) 135
== /2 (prop) 135
\== /2 (pred) 136
@< /2 (pred) 136
@=< /2 (pred) 137
@> /2 (pred) 137
@>= /2 (pred) 137
compare/3 (pred) 138
comparator/1 (regtype) 138
18.3 Known bugs and planned improvements (`term_compare`) 139

19 Basic predicates handling names of constants 141

19.1 Usage and interface (`atomic_basic`) 141
19.2 Documentation on exports (`atomic_basic`) 141
name/2 (pred) 141
atom_codes/2 (pred) 142
number_codes/2 (pred) 143
atom_number/2 (pred) 143
atom_number/3 (pred) 146
atom_length/2 (pred) 147
atom_concat/3 (pred) 147
sub_atom/4 (pred) 149
valid_base/1 (regtype) 149
19.3 Known bugs and planned improvements (`atomic_basic`) 149

20 Arithmetic 151

20.1 Usage and interface (`arithmetic`) 151
20.2 Documentation on exports (`arithmetic`) 151
is/2 (pred) 151
< /2 (pred) 153
=< /2 (pred) 153
> /2 (pred) 154
>= /2 (pred) 155
=:= /2 (pred) 155
=\= /2 (pred) 156
arithexpression/1 (regtype) 156
intexpression/1 (regtype) 158
20.3 Known bugs and planned improvements (`arithmetic`) 158

21 Basic file/stream handling 159

21.1 Usage and interface (`streams_basic`) 159
21.2 Documentation on exports (`streams_basic`) 159
open/3 (pred) 159
open/4 (pred) 160
open_option_list/1 (regtype) 160
close/1 (pred) 160
set_input/1 (pred) 161
current_input/1 (pred) 161
set_output/1 (pred) 161
current_output/1 (pred) 162
character_count/2 (pred) 162
line_count/2 (pred) 162
line_position/2 (pred) 163
flush_output/1 (pred) 163
flush_output/0 (pred) 163
clearerr/1 (pred) 163
current_stream/3 (pred) 163
stream_code/2 (pred) 164
absolute_file_name/2 (pred) 164
absolute_file_name/7 (pred) 165
pipe/2 (pred) 165
sourcename/1 (regtype) 165
stream/1 (regtype) 166
stream_alias/1 (regtype) 166
io_mode/1 (regtype) 167
atm_or_int/1 (regtype) 167
21.3 Documentation on multifiles (`streams_basic`) 167
file_search_path/2 (pred) 167
library_directory/1 (pred) 167
21.4 Known bugs and planned improvements (`streams_basic`) ... 168

22 Basic input/output 169

22.1 Usage and interface (`io_basic`) 169
22.2 Documentation on exports (`io_basic`) 169
get_code/2 (pred) 169
get_code/1 (pred) 169
get1_code/2 (pred) 170
get1_code/1 (pred) 170
peek_code/2 (pred) 170
peek_code/1 (pred) 171
skip_code/2 (pred) 171
skip_code/1 (pred) 171
skip_line/1 (pred) 171
skip_line/0 (pred) 171
put_code/2 (pred) 171
put_code/1 (pred) 172
nl/1 (pred) 172
nl/0 (pred) 172
tab/2 (pred) 173
tab/1 (pred) 173
code_class/2 (pred) 173
getct/2 (pred) 173
getct1/2 (pred) 174
display/2 (pred) 174

display/1 (pred) ... 175
displayq/2 (pred) ... 175
displayq/1 (pred) ... 175
22.3 Known bugs and planned improvements (io_basic) ... 175

23 Exception and Signal handling ... 177

23.1 Usage and interface (exceptions) ... 177
23.2 Documentation on exports (exceptions) ... 177
catch/3 (pred) ... 177
intercept/3 (pred) ... 177
throw/1 (pred) ... 178
send_signal/1 (pred) ... 178
send_silent_signal/1 (pred) ... 178
halt/0 (pred) ... 179
halt/1 (pred) ... 179
abort/0 (pred) ... 179
23.3 Known bugs and planned improvements (exceptions) ... 179

24 Changing system behaviour and various flags ... 181

24.1 Usage and interface (prolog_flags) ... 182
24.2 Documentation on exports (prolog_flags) ... 182
set_prolog_flag/2 (pred) ... 182
current_prolog_flag/2 (pred) ... 182
prolog_flag/3 (pred) ... 182
push_prolog_flag/2 (pred) ... 183
pop_prolog_flag/1 (pred) ... 183
set_ciao_flag/2 (pred) ... 184
current_ciao_flag/2 (pred) ... 184
ciao_flag/3 (pred) ... 184
push_ciao_flag/2 (pred) ... 184
pop_ciao_flag/1 (pred) ... 184
prompt/2 (pred) ... 184
gc/0 (pred) ... 185
nogc/0 (pred) ... 185
fileerrors/0 (pred) ... 185
nofileerrors/0 (pred) ... 185
24.3 Documentation on multifiles (prolog_flags) ... 186
define_flag/3 (pred) ... 186
24.4 Documentation on internals (prolog_flags) ... 186
set_prolog_flag/1 (pred) ... 186
24.5 Known bugs and planned improvements (prolog_flags) ... 186

25 Fast/concurrent update of facts 187

25.1 Usage and interface (`data_facts`) 187
25.2 Documentation on exports (`data_facts`) 187
asserta_fact/1 (pred) 187
asserta_fact/2 (pred) 187
assertz_fact/1 (pred) 188
assertz_fact/2 (pred) 188
current_fact/1 (pred) 188
current_fact/2 (pred) 189
retract_fact/1 (pred) 189
retractall_fact/1 (pred) 190
current_fact_nb/1 (pred) 190
retract_fact_nb/1 (pred) 190
close_predicate/1 (pred) 190
open_predicate/1 (pred) 191
set_fact/1 (pred) 191
erase/1 (pred) 191
reference/1 (regtype) 192
25.3 Documentation on internals (`data_facts`) 192
(data)/1 (decl) 192
(concurrent)/1 (decl) 192
25.4 Known bugs and planned improvements (`data_facts`) 192

26 Extending the syntax 193

26.1 Usage and interface (`syntax_extensions`) 193
26.2 Documentation on internals (`syntax_extensions`) 193
op/3 (decl) 193
new_declaration/1 (decl) 193
new_declaration/2 (decl) 193
load_compilation_module/1 (decl) 194
add_sentence_trans/2 (decl) 194
add_term_trans/2 (decl) 194
add_goal_trans/2 (decl) 195
add_clause_trans/2 (decl) 195
translation_predname/1 (regtype) 195

27 Message printing primitives 197

27.1 Usage and interface (`io_aux`) 197
27.2 Documentation on exports (`io_aux`) 197
message/2 (pred) 197
message_lns/4 (pred) 198
messages/1 (pred) 198
error/1 (pred) 198
warning/1 (pred) 198
note/1 (pred) 199
message/1 (pred) 199
debug/1 (pred) 199
inform_user/1 (pred) 199
display_string/1 (pred) 199
display_list/1 (pred) 199
display_term/1 (pred) 199
message_info/1 (regtype) 199
message_type/1 (regtype) 200
add_lines/4 (pred) 200
27.3 Known bugs and planned improvements (`io_aux`) 200

28 Attributed variables 201

28.1 Usage and interface (`attributes`) 201
28.2 Documentation on exports (`attributes`) 201
attach_attribute/2 (pred) 201
get_attribute/2 (pred) 201
update_attribute/2 (pred) 201
detach_attribute/1 (pred) 202
28.3 Documentation on multifiles (`attributes`) 202
verify_attribute/2 (pred) 202
combine_attributes/2 (pred) 202
28.4 Other information (`attributes`) 203
28.5 Known bugs and planned improvements (`attributes`) 203

29 Internal Runtime Information 205

29.1 Usage and interface (`system_info`) 205
29.2 Documentation on exports (`system_info`) 205
get_arch/1 (pred) 205
get_os/1 (pred) 205
get_platform/1 (pred) 206
get_debug/1 (pred) 206
get_eng_location/1 (pred) 206
get_ciao_ext/1 (pred) 206
get_exec_ext/1 (pred) 206
get_so_ext/1 (pred) 207
this_module/1 (pred) 207
current_module/1 (pred) 207
ciao_c_headers_dir/1 (pred) 207
ciao_lib_dir/1 (pred) 208
internal_module_id/1 (regtype) 208
29.3 Known bugs and planned improvements (`system_info`) 208

30 Conditional Compilation 209

30.1 Usage and interface (`condcomp_doc`) 209
30.2 Known bugs and planned improvements (`condcomp_doc`) 209

31 Other predicates and features defined by default 211

31.1 Usage and interface (`default_predicates`) 211
31.2 Documentation on exports (`default_predicates`) 211
op/3 (udreexp) 211
current_op/3 (udreexp) 211
append/3 (udreexp) 211
delete/3 (udreexp) 211
select/3 (udreexp) 211
nth/3 (udreexp) 212
last/2 (udreexp) 212
reverse/2 (udreexp) 212
length/2 (udreexp) 212
use_module/1 (udreexp) 212
use_module/2 (udreexp) 212
ensure_loaded/1 (udreexp) 212
(^)/2 (udreexp) 212
findnsols/5 (udreexp) 212
findnsols/4 (udreexp) 212

findall/4 (udreexp) . . . 212
findall/3 (udreexp) . . . 212
bagof/3 (udreexp) . . . 213
setof/3 (udreexp) . . . 213
wellformed_body/3 (udreexp) . . . 213
(data)/1 (udreexp) . . . 213
(dynamic)/1 (udreexp) . . . 213
current_predicate/2 (udreexp) . . . 213
current_predicate/1 (udreexp) . . . 213
clause/3 (udreexp) . . . 213
clause/2 (udreexp) . . . 213
abolish/1 (udreexp) . . . 213
retractall/1 (udreexp) . . . 213
retract/1 (udreexp) . . . 213
assert/2 (udreexp) . . . 214
assert/1 (udreexp) . . . 214
assertz/2 (udreexp) . . . 214
assertz/1 (udreexp) . . . 214
asserta/2 (udreexp) . . . 214
asserta/1 (udreexp) . . . 214
read_option/1 (udreexp) . . . 214
second_prompt/2 (udreexp) . . . 214
read_top_level/3 (udreexp) . . . 214
read_term/3 (udreexp) . . . 214
read_term/2 (udreexp) . . . 214
read/2 (udreexp) . . . 214
read/1 (udreexp) . . . 215
write_attribute/1 (udreexp) . . . 215
printable_char/1 (udreexp) . . . 215
prettyvars/1 (udreexp) . . . 215
numbervars/3 (udreexp) . . . 215
portray_clause/1 (udreexp) . . . 215
portray_clause/2 (udreexp) . . . 215
write_list1/1 (udreexp) . . . 215
print/1 (udreexp) . . . 215
print/2 (udreexp) . . . 215
write_canonical/1 (udreexp) . . . 215
write_canonical/2 (udreexp) . . . 215
writeq/1 (udreexp) . . . 216
writeq/2 (udreexp) . . . 216
write/1 (udreexp) . . . 216
write/2 (udreexp) . . . 216
write_option/1 (udreexp) . . . 216
write_term/2 (udreexp) . . . 216
write_term/3 (udreexp) . . . 216
put_char/2 (udreexp) . . . 216
put_char/1 (udreexp) . . . 216
peek_char/2 (udreexp) . . . 216
peek_char/1 (udreexp) . . . 216
get_char/2 (udreexp) . . . 216
get_char/1 (udreexp) . . . 217
put_byte/2 (udreexp) . . . 217
put_byte/1 (udreexp) . . . 217
peek_byte/2 (udreexp) . . . 217
peek_byte/1 (udreexp) . . . 217
get_byte/2 (udreexp) . . . 217

get_byte/1 (udreexp) 217
char_codes/2 (udreexp) 217
number_chars/2 (udreexp) 217
atom_chars/2 (udreexp) 217
char_code/2 (udreexp) 217
unify_with_occurs_check/2 (udreexp) 217
sub_atom/5 (udreexp) 218
compound/1 (udreexp) 218
once/1 (udreexp) 218
format_control/1 (udreexp) 218
format_to_string/3 (udreexp) 218
sformat/3 (udreexp) 218
format/3 (udreexp) 218
format/2 (udreexp) 218
keypair/1 (udreexp) 218
keylist/1 (udreexp) 218
keysort/2 (udreexp) 218
sort/2 (udreexp) 218
between/3 (udreexp) 219
system_error_report/1 (udreexp) 219
replace_characters/4 (udreexp) 219
no_swapslash/3 (udreexp) 219
cyg2win/3 (udreexp) 219
winpath_c/3 (udreexp) 219
winpath/3 (udreexp) 219
winpath/2 (udreexp) 219
using_windows/0 (udreexp) 219
rename_file/2 (udreexp) 219
delete_directory/1 (udreexp) 219
delete_file/1 (udreexp) 219
set_exec_mode/2 (udreexp) 220
chmod/3 (udreexp) 220
chmod/2 (udreexp) 220
fmode/2 (udreexp) 220
modif_time0/2 (udreexp) 220
modif_time/2 (udreexp) 220
file_properties/6 (udreexp) 220
file_property/2 (udreexp) 220
file_exists/2 (udreexp) 220
file_exists/1 (udreexp) 220
mktemp_in_tmp/2 (udreexp) 220
mktemp/2 (udreexp) 220
directory_files/2 (udreexp) 221
wait/3 (udreexp) 221
exec/8 (udreexp) 221
exec/3 (udreexp) 221
exec/4 (udreexp) 221
popen_mode/1 (udreexp) 221
popen/3 (udreexp) 221
system/2 (udreexp) 221
system/1 (udreexp) 221
shell/2 (udreexp) 221
shell/1 (udreexp) 221
shell/0 (udreexp) 221
cd/1 (udreexp) 222
working_directory/2 (udreexp) 222

make_dirpath/1 (udreexp) 222
make_dirpath/2 (udreexp) 222
make_directory/1 (udreexp) 222
make_directory/2 (udreexp) 222
umask/2 (udreexp) 222
current_executable/1 (udreexp) 222
current_host/1 (udreexp) 222
get_address/2 (udreexp) 222
get_tmp_dir/1 (udreexp) 222
get_grnam/1 (udreexp) 222
get_pwnam/1 (udreexp) 223
get_gid/1 (udreexp) 223
get_uid/1 (udreexp) 223
get_pid/1 (udreexp) 223
file_dir_name/3 (udreexp) 223
extract_paths/2 (udreexp) 223
dir_path/2 (udreexp) 223
copy_file/3 (udreexp) 223
copy_file/2 (udreexp) 223
c_errno/1 (udreexp) 223
del_env/1 (udreexp) 223
set_env/2 (udreexp) 223
current_env/2 (udreexp) 224
setenvstr/2 (udreexp) 224
getenvstr/2 (udreexp) 224
datime_struct/1 (udreexp) 224
datime/9 (udreexp) 224
datime/1 (udreexp) 224
time/1 (udreexp) 224
pause/1 (udreexp) 224
garbage_collect/0 (udreexp) 224
current_atom/1 (udreexp) 224
predicate_property/3 (udreexp) 224
predicate_property/2 (udreexp) 224
time_option/1 (udreexp) 225
tick_option/1 (udreexp) 225
clockfreq_option/1 (udreexp) 225
memory_option/1 (udreexp) 225
garbage_collection_option/1 (udreexp) 225
symbol_option/1 (udreexp) 225
time_result/1 (udreexp) 225
memory_result/1 (udreexp) 225
gc_result/1 (udreexp) 225
symbol_result/1 (udreexp) 225
new_atom/1 (udreexp) 225
tick_result/1 (udreexp) 225
clockfreq_result/1 (udreexp) 226
statistics/2 (udreexp) 226
statistics/0 (udreexp) 226
close_file/1 (udreexp) 226
told/0 (udreexp) 226
telling/1 (udreexp) 226
tell/1 (udreexp) 226
seen/0 (udreexp) 226
seeing/1 (udreexp) 226
see/1 (udreexp) 226

current_key/2 (udreexp) 226
recorded/3 (udreexp) 226
recordz/3 (udreexp) 227
recorda/3 (udreexp) 227
ttydisplay_string/1 (udreexp) 227
ttyskipeol/0 (udreexp) 227
ttydisplayq/1 (udreexp) 227
ttydisplay/1 (udreexp) 227
ttyflush/0 (udreexp) 227
ttytab/1 (udreexp) 227
ttyskip/1 (udreexp) 227
ttyput/1 (udreexp) 227
ttynl/0 (udreexp) 227
ttyget1/1 (udreexp) 227
ttyget/1 (udreexp) 227

PART III - ISO-Prolog library (iso) 229

32 ISO-Prolog package 231
32.1 Usage and interface (`iso_doc`) 231

33 All solutions predicates 233
33.1 Usage and interface (`aggregates`) 233
33.2 Documentation on exports (`aggregates`) 233
setof/3 (pred) 233
bagof/3 (pred) 234
findall/3 (pred) 234
findall/4 (pred) 235
findnsols/4 (pred) 235
findnsols/5 (pred) 236
(^)/2 (pred) 236
33.3 Known bugs and planned improvements (`aggregates`) 237

34 Dynamic predicates 239
34.1 Usage and interface (`dynamic_rt`) 239
34.2 Documentation on exports (`dynamic_rt`) 239
asserta/1 (pred) 239
asserta/2 (pred) 240
assertz/1 (pred) 240
assertz/2 (pred) 240
assert/1 (pred) 240
assert/2 (pred) 241
retract/1 (pred) 241
retractall/1 (pred) 241
abolish/1 (pred) 242
clause/2 (pred) 242
mfclause/2 (pred) 242
current_predicate/1 (pred) 242
current_predicate/2 (pred) 243
(dynamic)/1 (pred) 243
(data)/1 (pred) 243
erase/1 (pred) 244
wellformed_body/3 (pred) 244
34.3 Documentation on multifiles (`dynamic_rt`) 244

do_on_abolish/1 (pred) 244
34.4 Known bugs and planned improvements (dynamic_rt) 244

35 Term input 245
35.1 Usage and interface (read) 245
35.2 Documentation on exports (read) 245
read/1 (pred) 245
read/2 (pred) 245
read_term/2 (pred) 246
read_term/3 (pred) 246
read_top_level/3 (pred) 246
second_prompt/2 (pred) 246
read_option/1 (regtype) 246
35.3 Documentation on multifiles (read) 247
define_flag/3 (pred) 247
35.4 Known bugs and planned improvements (read) 247

36 Term output 249
36.1 Usage and interface (write) 249
36.2 Documentation on exports (write) 249
write_term/3 (pred) 249
write_term/2 (pred) 249
write_option/1 (prop) 250
write/2 (pred) 250
write/1 (pred) 251
writeq/2 (pred) 251
writeq/1 (pred) 251
write_canonical/2 (pred) 251
write_canonical/1 (pred) 252
print/2 (pred) 252
print/1 (pred) 252
write_list1/1 (pred) 252
portray_clause/2 (pred) 252
portray_clause/1 (pred) 253
numbervars/3 (pred) 253
prettyvars/1 (pred) 253
printable_char/1 (pred) 253
write_attribute/1 (pred) 253
36.3 Documentation on multifiles (write) 254
define_flag/3 (pred) 254
portray_attribute/2 (pred) 254
portray/1 (pred) 254
36.4 Known bugs and planned improvements (write) 254

37 Defining operators 255
37.1 Usage and interface (operators) 255
37.2 Documentation on exports (operators) 255
op/3 (pred) 255
current_op/3 (pred) 256
current_prefixop/3 (pred) 256
current_infixop/4 (pred) 256
current_postfixop/3 (pred) 257
standard_ops/0 (pred) 257

38 The Iso Byte Char module 259

38.1 Usage and interface (`iso_byte_char`) 259
38.2 Documentation on exports (`iso_byte_char`) 259
char_code/2 (pred) 259
atom_chars/2 (pred) 259
number_chars/2 (pred) 260
char_codes/2 (pred) 260
get_byte/1 (pred) 261
get_byte/2 (pred) 261
peek_byte/1 (pred) 261
peek_byte/2 (pred) 261
put_byte/1 (pred) 262
put_byte/2 (pred) 262
get_char/1 (pred) 262
get_char/2 (pred) 262
peek_char/1 (pred) 263
peek_char/2 (pred) 263
put_char/1 (pred) 263
put_char/2 (pred) 263

39 Miscellaneous ISO Prolog predicates 265

39.1 Usage and interface (`iso_misc`) 265
39.2 Documentation on exports (`iso_misc`) 265
once/1 (pred) 265
compound/1 (pred) 265
sub_atom/5 (pred) 265
unify_with_occurs_check/2 (pred) 266
39.3 Known bugs and planned improvements (`iso_misc`) 266

40 Incomplete ISO Prolog predicates 267

40.1 Usage and interface (`iso_incomplete`) 267
40.2 Documentation on exports (`iso_incomplete`) 267
close/2 (pred) 267
stream_property/2 (pred) 267

PART IV - Classic Prolog library (classic) 269

41 Definite clause grammars 271

41.1 Usage and interface (`dcg_doc`) 273

42 Definite Clause Grammars (expansion) 275

42.1 Usage and interface (`dcg_tr`) 275
42.2 Documentation on exports (`dcg_tr`) 275
phrase/2 (pred) 275
phrase/3 (pred) 275
dcg_translation/2 (pred) 275
42.3 Known bugs and planned improvements (`dcg_tr`) 275

43 Formatted output 277

43.1 Usage and interface (`format`) 277
43.2 Documentation on exports (`format`) 277
format/2 (pred) 277
format/3 (pred) 278
sformat/3 (pred) 278
format_to_string/3 (pred) 278
format_control/1 (regtype) 279
43.3 Known bugs and planned improvements (`format`) 281

44 List processing 283

44.1 Usage and interface (`lists`) 283
44.2 Documentation on exports (`lists`) 283
nonsingle/1 (pred) 283
append/3 (pred) 283
reverse/2 (pred) 286
reverse/3 (pred) 286
delete/3 (pred) 287
delete_non_ground/3 (pred) 287
select/3 (pred) 288
length/2 (pred) 288
nth/3 (pred) 289
add_after/4 (pred) 290
add_before/4 (pred) 290
list1/2 (prop) 290
dlist/3 (pred) 290
list_concat/2 (pred) 291
list_insert/2 (pred) 291
insert_last/3 (pred) 291
contains_ro/2 (pred) 291
contains1/2 (pred) 291
nocontainsx/2 (pred) 291
last/2 (pred) 291
list_lookup/3 (pred) 292
list_lookup/4 (pred) 292
intset_insert/3 (pred) 292
intset_delete/3 (pred) 292
intset_in/2 (pred) 292
intset_sequence/3 (pred) 292
intersection/3 (pred) 292
union/3 (pred) 293
difference/3 (pred) 293
sublist/2 (prop) 293
subordlist/2 (prop) 293
equal_lists/2 (pred) 293
list_to_list_of_lists/2 (pred) 294
powerset/2 (pred) 294
cross_product/2 (pred) 294
sequence_to_list/2 (pred) 294
list_of_lists/1 (regtype) 294

45 Sorting lists 295

45.1 Usage and interface (`sort`) 295
45.2 Documentation on exports (`sort`) 295
sort/2 (pred) 295
keysort/2 (pred) 296
keylist/1 (regtype) 296
keypair/1 (regtype) 296
45.3 Known bugs and planned improvements (`sort`) 296

46 compiler (library) 297

46.1 Usage and interface (`compiler`) 297
46.2 Documentation on exports (`compiler`) 297
make_po/1 (pred) 297
make_wam/1 (pred) 297
ensure_loaded/1 (pred) 297
ensure_loaded/2 (pred) 297
use_module/1 (pred) 297
use_module/2 (pred) 297
use_module/3 (pred) 297
unload/1 (pred) 298
set_debug_mode/1 (pred) 298
set_nodebug_mode/1 (pred) 298
set_debug_module/1 (pred) 298
set_nodebug_module/1 (pred) 298
set_debug_module_source/1 (pred) 298
mode_of_module/2 (pred) 298
module_of/2 (pred) 298

47 Enumeration of integers inside a range 299

47.1 Usage and interface (`between`) 299
47.2 Documentation on exports (`between`) 299
between/3 (pred) 299

48 Operating system utilities 301

48.1 Usage and interface (`system`) 301
48.2 Documentation on exports (`system`) 301
pause/1 (pred) 301
time/1 (pred) 301
datime/1 (pred) 302
datime/9 (pred) 302
datime_struct/1 (regtype) 304
getenvstr/2 (pred) 304
setenvstr/2 (pred) 304
current_env/2 (pred) 304
set_env/2 (pred) 304
del_env/1 (pred) 305
c_errno/1 (pred) 305
copy_file/2 (pred) 305
copy_file/3 (pred) 305
dir_path/2 (pred) 305
extract_paths/2 (pred) 306
file_dir_name/3 (pred) 306
get_pid/1 (pred) 306
get_uid/1 (pred) 307

get_gid/1 (pred) . . . 307
get_pwnam/1 (pred) . . . 307
get_grnam/1 (pred) . . . 307
get_tmp_dir/1 (pred) . . . 307
get_address/2 (pred) . . . 308
current_host/1 (pred) . . . 308
current_executable/1 (pred) . . . 308
umask/2 (pred) . . . 308
make_directory/2 (pred) . . . 309
make_directory/1 (pred) . . . 309
make_dirpath/2 (pred) . . . 309
make_dirpath/1 (pred) . . . 309
working_directory/2 (pred) . . . 309
cd/1 (pred) . . . 310
shell/0 (pred) . . . 310
shell/1 (pred) . . . 310
shell/2 (pred) . . . 310
system/1 (pred) . . . 311
system/2 (pred) . . . 311
popen/3 (pred) . . . 311
popen_mode/1 (regtype) . . . 311
exec/4 (pred) . . . 312
exec/3 (pred) . . . 312
exec/8 (pred) . . . 312
wait/3 (pred) . . . 313
directory_files/2 (pred) . . . 313
mktemp/2 (pred) . . . 314
mktemp_in_tmp/2 (pred) . . . 314
file_exists/1 (pred) . . . 314
file_exists/2 (pred) . . . 314
file_property/2 (pred) . . . 315
file_properties/6 (pred) . . . 315
modif_time/2 (pred) . . . 316
modif_time0/2 (pred) . . . 316
fmode/2 (pred) . . . 316
chmod/2 (pred) . . . 317
chmod/3 (pred) . . . 317
set_exec_mode/2 (pred) . . . 317
delete_file/1 (pred) . . . 317
delete_directory/1 (pred) . . . 318
rename_file/2 (pred) . . . 318
using_windows/0 (pred) . . . 318
winpath/2 (pred) . . . 318
winpath/3 (pred) . . . 319
winpath_c/3 (pred) . . . 319
cyg2win/3 (pred) . . . 319
no_swapslash/3 (pred) . . . 320
replace_characters/4 (pred) . . . 320
system_error_report/1 (pred) . . . 320
48.3 Documentation on multifiles (`system`) . . . 320
define_flag/3 (pred) . . . 320
48.4 Known bugs and planned improvements (`system`) . . . 321

49 Prolog system internal predicates 323

49.1 Usage and interface (`prolog_sys`) 323
49.2 Documentation on exports (`prolog_sys`) 323
statistics/0 (pred) 323
statistics/2 (pred) 323
clockfreq_result/1 (regtype) 325
tick_result/1 (regtype) 325
new_atom/1 (pred) 325
symbol_result/1 (regtype) 326
gc_result/1 (regtype) 326
memory_result/1 (regtype) 326
time_result/1 (regtype) 326
symbol_option/1 (regtype) 326
garbage_collection_option/1 (regtype) 326
memory_option/1 (regtype) 326
clockfreq_option/1 (regtype) 326
tick_option/1 (regtype) 327
time_option/1 (regtype) 327
predicate_property/2 (pred) 327
predicate_property/3 (pred) 327
current_atom/1 (pred) 327
garbage_collect/0 (pred) 327
49.3 Known bugs and planned improvements (`prolog_sys`) 328

50 DEC-10 Prolog file IO 329

50.1 Usage and interface (`dec10_io`) 329
50.2 Documentation on exports (`dec10_io`) 329
see/1 (pred) 329
seeing/1 (pred) 329
seen/0 (pred) 329
tell/1 (pred) 329
telling/1 (pred) 329
told/0 (pred) 329
close_file/1 (pred) 330

51 Quintus-like internal database 331

51.1 Usage and interface (`old_database`) 331
51.2 Documentation on exports (`old_database`) 331
recorda/3 (pred) 331
recordz/3 (pred) 331
recorded/3 (pred) 332
current_key/2 (pred) 332

52 ttyout (library) 333
52.1 Usage and interface (`ttyout`) 333
52.2 Documentation on exports (`ttyout`) 333
ttyget/1 (pred) 333
ttyget1/1 (pred) 333
ttynl/0 (pred) 333
ttyput/1 (pred) 333
ttyskip/1 (pred) 333
ttytab/1 (pred) 333
ttyflush/0 (pred) 334
ttydisplay/1 (pred) 334
ttydisplayq/1 (pred) 334
ttyskipeol/0 (pred) 334
ttydisplay_string/1 (pred) 334

53 Enabling operators at run-time 335
53.1 Usage and interface (`runtime_ops_doc`) 335

PART V - Assertions, Properties, Types, Modes, Comments (assertions) 337

54 The Ciao assertion package 339
54.1 More info 339
54.2 Some attention points 339
54.3 Usage and interface (`assertions_doc`) 340
54.4 Documentation on new declarations (`assertions_doc`) 340
(pred)/1 (decl) 340
(pred)/2 (decl) 341
(texec)/1 (decl) 341
(texec)/2 (decl) 341
(calls)/1 (decl) 341
(calls)/2 (decl) 341
(success)/1 (decl) 342
(success)/2 (decl) 342
(test)/1 (decl) 342
(test)/2 (decl) 342
(comp)/1 (decl) 343
(comp)/2 (decl) 343
(prop)/1 (decl) 343
(prop)/2 (decl) 344
(entry)/1 (decl) 344
(exit)/1 (decl) 344
(exit)/2 (decl) 344
(modedef)/1 (decl) 345
(decl)/1 (decl) 345
(decl)/2 (decl) 345
doc/2 (decl) 345
comment/2 (decl) 346
54.5 Documentation on exports (`assertions_doc`) 346
check/1 (pred) 346
trust/1 (pred) 346
true/1 (pred) 347
false/1 (pred) 347

55 Types and properties related to assertions .. 349

55.1 Usage and interface (`assertions_props`) 349
55.2 Documentation on exports (`assertions_props`) 349
assrt_body/1 (regtype) 349
head_pattern/1 (prop) 350
complex_arg_property/1 (regtype) 350
property_conjunction/1 (regtype) 351
property_starterm/1 (regtype) 351
complex_goal_property/1 (regtype) 351
nabody/1 (prop) 352
dictionary/1 (regtype) 352
c_assrt_body/1 (regtype) 352
s_assrt_body/1 (regtype) 352
g_assrt_body/1 (regtype) 353
assrt_status/1 (regtype) 353
assrt_type/1 (regtype) 354
predfunctor/1 (regtype) 354
propfunctor/1 (regtype) 354
docstring/1 (prop) 354

56 Declaring regular types 355

56.1 Defining properties 355
56.2 Usage and interface (`regtypes_doc`) 357
56.3 Documentation on new declarations (`regtypes_doc`) 358
(regtype)/1 (decl) 358
(regtype)/2 (decl) 358

57 Properties which are native to analyzers 359

57.1 Usage and interface (`native_props`) 359
57.2 Documentation on exports (`native_props`) 359
clique/1 (prop) 359
clique_1/1 (prop) 359
constraint/1 (prop) 360
covered/1 (prop) 360
covered/2 (prop) 360
exception/1 (prop) 360
exception/2 (prop) 360
fails/1 (prop) 361
finite_solutions/1 (prop) 361
have_choicepoints/1 (prop) 361
indep/1 (prop) 361
indep/2 (prop) 361
is_det/1 (prop) 361
linear/1 (prop) 362
mshare/1 (prop) 362
mut_exclusive/1 (prop) 362
no_choicepoints/1 (prop) 362
no_exception/1 (prop) 363
no_exception/2 (prop) 363
no_signal/1 (prop) 363
no_signal/2 (prop) 363
non_det/1 (prop) 363
nonground/1 (prop) 363
not_covered/1 (prop) 363
not_fails/1 (prop) 364

not_mut_exclusive/1 (prop) . . . 364
num_solutions/2 (prop) . . . 364
solutions/2 (prop) . . . 364
possibly_fails/1 (prop) . . . 365
possibly_nondet/1 (prop) . . . 365
relations/2 (prop) . . . 365
sideff_hard/1 (prop) . . . 365
sideff_pure/1 (prop) . . . 365
sideff_soft/1 (prop) . . . 366
signal/1 (prop) . . . 366
signal/2 (prop) . . . 366
signals/2 (prop) . . . 366
size/2 (prop) . . . 366
size/3 (prop) . . . 366
size_lb/2 (prop) . . . 366
size_o/2 (prop) . . . 366
size_ub/2 (prop) . . . 367
size_metric/3 (prop) . . . 367
size_metric/4 (prop) . . . 367
steps/2 (prop) . . . 367
steps_lb/2 (prop) . . . 367
steps_o/2 (prop) . . . 367
steps_ub/2 (prop) . . . 368
tau/1 (prop) . . . 368
terminates/1 (prop) . . . 368
test_type/2 (prop) . . . 368
throws/2 (prop) . . . 368
user_output/2 (prop) . . . 369
instance/2 (prop) . . . 369
57.3 Known bugs and planned improvements (`native_props`) 369

58 ISO-Prolog modes . . . 371

58.1 Usage and interface (`isomodes_doc`) . . . 371
58.2 Documentation on new modes (`isomodes_doc`) . . . 371
(+)/1 (modedef) . . . 371
(@)/1 (modedef) . . . 371
(-)/1 (modedef) . . . 371
(?)/1 (modedef) . . . 371
* /1 (modedef) . . . 371
(+)/2 (modedef) . . . 371
(@)/2 (modedef) . . . 372
(-)/2 (modedef) . . . 372
(?)/2 (modedef) . . . 372
* /2 (modedef) . . . 372

59 Classical Prolog modes 373

59.1 Usage and interface (`basicmodes_doc`) 373
59.2 Documentation on new modes (`basicmodes_doc`) 373
(+)/1 (modedef) 373
(-)/1 (modedef) 373
(?)/1 (modedef) 373
(@)/1 (modedef) 373
in/1 (modedef) 374
out/1 (modedef) 374
go/1 (modedef) 374
(+)/2 (modedef) 374
(-)/2 (modedef) 374
(?)/2 (modedef) 374
(@)/2 (modedef) 375
in/2 (modedef) 375
out/2 (modedef) 375
go/2 (modedef) 375

60 Run-time checking of assertions 377

60.1 Usage and interface (`rtchecks_doc`) 378

61 Unit Testing Library 379

61.1 Additional notes 379
61.2 Usage and interface (`unittest_doc`) 380

PART VI - Ciao library miscellanea 381

62 Library Paths for Ciao Components 383

62.1 Usage and interface (`ciaopaths_doc`) 383
62.2 Known bugs and planned improvements (`ciaopaths_doc`) ... 383

63 Analytic benchmarks 385

63.1 Testing Calls 385
63.2 Testing non-deterministic behavior 385
63.3 Testing environment handling 385
63.4 Testing indexing mechanisms 386
63.5 Testing unification 386
63.6 Testing dereferencing 386
63.7 Testing the cut 387
63.8 Assorted small programs 387
63.9 Usage and interface (`ecrc`) 388
63.10 Documentation on exports (`ecrc`) 388
main/1 (pred) 388
benchmark_usage/1 (regtype) 388
just_benchmarks/0 (pred) 389
generate_human_file/0 (pred) 389
generate_machine_file/0 (pred) 389
send_info_to_developers/0 (pred) 389
arithm_average/2 (pred) 390
geom_average/2 (pred) 390
63.11 Known bugs and planned improvements (`ecrc`) 390

64 Parse and return command-line options 391
64.1 Usage and interface (`getopts`) 391
64.2 Documentation on exports (`getopts`) 391
getopts/4 (pred) 391
cl_option/2 (pred) 392
64.3 Documentation on internals (`getopts`) 393
spec/1 (regtype) 393

65 llists (library) 395
65.1 Usage and interface (`llists`) 395
65.2 Documentation on exports (`llists`) 395
append/2 (pred) 395
flatten/2 (pred) 395
collect_singletons/2 (pred) 395
transpose/2 (pred) 396

66 Structured stream handling 397
66.1 Usage and interface (`streams`) 397
66.2 Documentation on exports (`streams`) 397
open_null_stream/1 (pred) 397
open_input/2 (pred) 397
close_input/1 (pred) 397
open_output/2 (pred) 397
close_output/1 (pred) 398

67 Dictionaries 399
67.1 Usage and interface (`dict`) 399
67.2 Documentation on exports (`dict`) 399
dictionary/1 (regtype) 399
dictionary/5 (pred) 399
dic_node/2 (pred) 399
dic_lookup/3 (pred) 399
dic_lookup/4 (pred) 400
dic_get/3 (pred) 400
dic_replace/4 (pred) 400
old_or_new/1 (regtype) 400
non_empty_dictionary/1 (regtype) 400
67.3 Known bugs and planned improvements (`dict`) 400

68 String processing 401
68.1 Usage and interface (`strings`) 401
68.2 Documentation on exports (`strings`) 401
get_line/2 (pred) 401
get_line/1 (pred) 401
line/1 (regtype) 401
write_string/2 (pred) 402
write_string/1 (pred) 402
whitespace/2 (pred) 402
whitespace0/2 (pred) 402
string/3 (pred) 403

69 Printing status and error messages 405

69.1 Usage and interface (`messages`) 405
69.2 Documentation on exports (`messages`) 405
error_message/1 (pred) 405
error_message/2 (pred) 405
error_message/3 (pred) 405
warning_message/1 (pred) 406
warning_message/2 (pred) 406
warning_message/3 (pred) 406
note_message/1 (pred) 407
note_message/2 (pred) 407
note_message/3 (pred) 407
simple_message/1 (pred) 407
simple_message/2 (pred) 407
optional_message/2 (pred) 408
optional_message/3 (pred) 408
debug_message/1 (pred) 408
debug_message/2 (pred) 408
debug_goal/2 (pred) 408
debug_goal/3 (pred) 409
show_message/2 (pred) 409
show_message/3 (pred) 409
show_message/4 (pred) 409
message_t/1 (regtype) 410
location_t/1 (udreexp) 410
69.3 Documentation on multifiles (`messages`) 410
issue_debug_messages/1 (pred) 410
69.4 Known bugs and planned improvements (`messages`) 410

70 Accessing and redirecting the stream aliases 411

70.1 Usage and interface (`io_alias_redirection`) 411
70.2 Documentation on exports (`io_alias_redirection`) 411
set_stream/3 (pred) 411
get_stream/2 (pred) 411

71 Atom to term conversion 413

71.1 Usage and interface (`atom2term`) 413
71.2 Documentation on exports (`atom2term`) 413
atom2term/2 (pred) 413
string2term/2 (pred) 414
parse_term/3 (pred) 414
parse_term/4 (pred) 415
71.3 Known bugs and planned improvements (`atom2term`) 416

72 ctrlcclean (library) 417

72.1 Usage and interface (`ctrlcclean`) 417
72.2 Documentation on exports (`ctrlcclean`) 417
ctrlc_clean/1 (pred) 417
delete_on_ctrlc/2 (pred) 417
ctrlcclean/0 (pred) 417

73 errhandle (library) 419
73.1 Usage and interface (errhandle) 419
73.2 Documentation on exports (errhandle) 419
error_protect/1 (pred) 419
handle_error/2 (pred) 419

74 Fast reading and writing of terms 421
74.1 Usage and interface (fastrw) 421
74.2 Documentation on exports (fastrw) 421
fast_read/1 (pred) 421
fast_write/1 (pred) 421
fast_read/2 (pred) 421
fast_write/2 (pred) 421
fast_write_to_string/3 (pred) 422
74.3 Known bugs and planned improvements (fastrw) 422

75 File name manipulation 423
75.1 Usage and interface (filenames) 423
75.2 Documentation on exports (filenames) 423
no_path_file_name/2 (pred) 423
file_directory_base_name/3 (pred) 424
file_name_extension/3 (pred) 424
basename/2 (pred) 425
atom_or_str/1 (regtype) 425
extension/2 (pred) 425

76 Symbolic filenames 427
76.1 Usage and interface (symfnames) 427
76.2 Documentation on exports (symfnames) 427
open/3 (pred) 427
76.3 Documentation on multifiles (symfnames) 428
alias_file/1 (pred) 428
file_alias/2 (pred) 428
76.4 Other information (symfnames) 428

77 File I/O utilities 429
77.1 Usage and interface (file_utils) 429
77.2 Documentation on exports (file_utils) 429
file_terms/2 (pred) 429
copy_stdout/1 (pred) 429
file_to_string/2 (pred) 429
file_to_string/3 (pred) 430
string_to_file/2 (pred) 430
stream_to_string/2 (pred) 430
stream_to_string/3 (pred) 431
output_to_file/2 (pred) 431

78 File locks 433
78.1 Usage and interface (file_locks) 433
78.2 Documentation on exports (file_locks) 433
lock_file/3 (pred) 433
unlock_file/2 (pred) 433
78.3 Known bugs and planned improvements (file_locks) 433

79 Lists and conjunctions and disjunctions 435

79.1 Usage and interface (`formulae`) 435
79.2 Documentation on exports (`formulae`) 435
list_to_conj/3 (pred) 435
list_to_conj/2 (pred) 435
conj_to_list/2 (pred) 436
list_to_disj/2 (pred) 436
disj_to_list/2 (pred) 437
conj_to_llist/2 (pred) 437
llist_to_conj/2 (pred) 437
disj_to_llist/2 (pred) 438
llist_to_disj/2 (pred) 438
body2list/2 (pred) 438
asbody_to_conj/2 (pred) 438
assert_body_type/1 (prop) 438
conj_disj_type/1 (regtype) 438
t_conj/1 (regtype) 438
t_disj/1 (regtype) 439
list_to_disj2/2 (pred) 439

80 Term manipulation utilities 441

80.1 Usage and interface (`terms`) 441
80.2 Documentation on exports (`terms`) 441
term_size/2 (pred) 441
copy_args/3 (pred) 441
arg/2 (pred) 442
atom_concat/2 (pred) 442

81 Term checking utilities 445

81.1 Usage and interface (`terms_check`) 445
81.2 Documentation on exports (`terms_check`) 445
ask/2 (pred) 445
instance/2 (prop) 445
variant/2 (pred) 445
most_general_instance/3 (pred) 445
most_specific_generalization/3 (pred) 445
81.3 Other information (`terms_check`) 446
81.4 Known bugs and planned improvements (`terms_check`) 446

82 Sets of variables in terms 447

82.1 Usage and interface (`terms_vars`) 447
82.2 Documentation on exports (`terms_vars`) 447
varset/2 (pred) 447
intersect_vars/3 (pred) 447
member_var/2 (pred) 447
diff_vars/3 (pred) 447
varsbag/3 (pred) 447
varset_in_args/2 (pred) 447

83 Cyclic terms handling 449

83.1 Usage and interface (`cyclic_terms`) 449
83.2 Documentation on exports (`cyclic_terms`) 449
cyclic_term/1 (pred) 449
acyclic_term/1 (pred) 449
uncycle_term/2 (pred) 449
recycle_term/2 (pred) 449

84 A simple pretty-printer for Ciao programs .. 451

84.1 Usage and interface (`pretty_print`) 451
84.2 Documentation on exports (`pretty_print`) 451
pretty_print/2 (pred) 451
pretty_print/3 (pred) 451
pretty_print/4 (pred) 451
84.3 Documentation on internals (`pretty_print`) 452
clauses/1 (regtype) 452
clause/1 (regtype) 452
clterm/1 (regtype) 452
body/1 (regtype) 452
flag/1 (regtype) 452
84.4 Known bugs and planned improvements (`pretty_print`) 453

85 Pretty-printing assertions 455

85.1 Usage and interface (`assrt_write`) 455
85.2 Documentation on exports (`assrt_write`) 455
write_assertion/6 (pred) 455
write_assertion/7 (pred) 455
write_assertion_as_comment/6 (pred) 455
write_assertion_as_comment/7 (pred) 456
write_assertion_as_double_comment/6 (pred) 456
write_assertion_as_double_comment/7 (pred) 456

86 The Ciao library browser 457

86.1 Usage and interface (`librowser`) 457
86.2 Documentation on exports (`librowser`) 458
update/0 (pred) 458
browse/2 (pred) 458
where/1 (pred) 458
describe/1 (pred) 459
system_lib/1 (pred) 459
apropos/1 (pred) 459
86.3 Documentation on internals (`librowser`) 460
apropos_spec/1 (regtype) 460

87 Code translation utilities 461

87.1 Usage and interface (`expansion_tools`) 461
87.2 Documentation on exports (`expansion_tools`) 461
imports_meta_pred/3 (pred) 461
body_expander/6 (pred) 461
arg_expander/6 (pred) 462
87.3 Documentation on internals (`expansion_tools`) 462
expander_pred/1 (prop) 462
87.4 Known bugs and planned improvements (`expansion_tools`) 463

88 Low-level concurrency/multithreading primitives 465

88.1 Usage and interface (`concurrency`) 465
88.2 Documentation on exports (`concurrency`) 465
eng_call/4 (pred) 465
eng_call/3 (pred) 466
eng_backtrack/2 (pred) 466
eng_cut/1 (pred) 466
eng_release/1 (pred) 467
eng_wait/1 (pred) 467
eng_kill/1 (pred) 467
eng_killothers/0 (pred) 467
eng_self/1 (pred) 467
goal_id/1 (pred) 468
eng_goal_id/1 (pred) 468
eng_status/0 (pred) 468
lock_atom/1 (pred) 468
unlock_atom/1 (pred) 468
atom_lock_state/2 (pred) 469
(concurrent)/1 (pred) 469
88.3 Known bugs and planned improvements (`concurrency`) 470

89 All solutions concurrent predicates 471

89.1 Usage and interface (`conc_aggregates`) 471
89.2 Documentation on exports (`conc_aggregates`) 471
findall/3 (pred) 471
setof/3 (pred) 471
bagof/3 (pred) 472
89.3 Known bugs and planned improvements (`conc_aggregates`) 472

90 The socket interface 473

90.1 Usage and interface (`sockets`) 473
90.2 Documentation on exports (`sockets`) 473
connect_to_socket_type/4 (pred) 473
connect_to_socket/3 (pred) 474
bind_socket/3 (pred) 474
socket_accept/2 (pred) 474
select_socket/5 (pred) 475
socket_send/2 (pred) 475
socket_recv_code/3 (pred) 475
socket_recv/2 (pred) 476
socket_shutdown/2 (pred) 476
hostname_address/2 (pred) 476
socket_type/1 (regtype) 477
shutdown_type/1 (regtype) 477

91 Sockets I/O 479

91.1 Usage and interface (`sockets_io`) 479
91.2 Documentation on exports (`sockets_io`) 479
serve_socket/3 (pred) 479
safe_write/2 (pred) 479

92 The Ciao Make Package 481

92.1 Usage and interface (`make_doc`) 481
92.2 Other information (`make_doc`) 481
92.2.1 The Dependency Rules 481
92.2.2 Specifying Paths 483
92.2.3 Documenting Rules 483
92.2.4 An Example of a Makefile 483

93 Predicates Available When Using The Make Package 487

93.1 Usage and interface (`make_rt`) 487
93.2 Documentation on exports (`make_rt`) 487
make/1 (pred) 487
target/1 (regtype) 487
make_option/1 (pred) 488
verbose_message/1 (pred) 488
verbose_message/2 (pred) 488
dot_concat/2 (pred) 488
call_unknown/1 (pred) 488
all_values/2 (pred) 488
get_value/2 (pred) 488
get_value_def/3 (pred) 488
get_all_values/2 (pred) 488
name_value/2 (pred) 489
set_name_value/2 (pred) 489
cp_name_value/2 (pred) 489
get_name_value/3 (pred) 489
get_name_value_string/3 (pred) 489
add_name_value/2 (pred) 489
del_name_value/1 (pred) 489
check_var_exists/1 (pred) 489
find_file/2 (pred) 489
vpath/1 (pred) 490
add_vpath/1 (pred) 490
vpath_mode/3 (pred) 490
add_vpath_mode/3 (pred) 490
bold_message/1 (pred) 490
bold_message/2 (pred) 490
normal_message/2 (pred) 490
bolder_message/1 (pred) 490
bolder_message/2 (pred) 490
newer/2 (pred) 490
register_module/1 (pred) 490
unregister_module/1 (pred) 490
push_name_value/3 (pred) 491
pop_name_value/1 (pred) 491
push_active_config/1 (pred) 491
pop_active_config/0 (pred) 491
get_active_config/1 (pred) 491
dyn_load_cfg_module_into_make/1 (pred) 491
get_settings_nvalue/1 (pred) 491
apply_vpath_mode/4 (pred) 491
get_name/2 (pred) 492
up_to_date/2 (udreexp) 492
93.3 Known bugs and planned improvements (`make_rt`) 492

94 Additional operating system utilities 493
94.1 Usage and interface (`system_extra`) 493
94.2 Documentation on exports (`system_extra`) 493
del_dir_if_empty/1 (pred) 493
move_files/2 (pred) 493
move_file/2 (pred) 494
copy_files/2 (pred) 494
copy_files/3 (pred) 494
copy_files_nofail/3 (pred) 494
cat/2 (pred) 494
cat_append/2 (pred) 494
symbolic_link/2 (pred) 494
symbolic_link/3 (pred) 494
delete_files/1 (pred) 495
del_files_nofail/1 (pred) 495
del_file_nofail/1 (pred) 495
del_file_nofail/2 (pred) 495
del_endings_nofail/2 (pred) 495
ls/3 (pred) 495
ls/2 (pred) 495
filter_alist_pattern/3 (pred) 495
(-)/1 (pred) 496
(–)/1 (pred) 496
do/2 (pred) 496
do/3 (pred) 496
do/4 (pred) 496
do/5 (pred) 497
try_finally/3 (pred) 497
set_owner/2 (pred) 497
readf/2 (pred) 497
datime_string/1 (pred) 498
datime_string/2 (pred) 498
datime_atom/1 (pred) 498
datime_atom/2 (pred) 498
no_tr_nl/2 (pred) 498
replace_strings/3 (pred) 498
replace_strings_in_file/3 (pred) 498
writef/3 (pred) 498
writef/2 (pred) 498
add_suffix/3 (pred) 498
add_preffix/3 (pred) 498
writef_list/3 (pred) 498
writef_list/2 (pred) 499
etags/2 (pred) 499
any_to_term/2 (pred) 499
touch/1 (pred) 499
get_perms/2 (pred) 499
set_perms/2 (pred) 499
set_exec_perms/2 (pred) 499
mkdir_perm/2 (pred) 499
convert_permissions/2 (pred) 499
convert_permissions/4 (pred) 499
execute_permissions/2 (pred) 499
execute_permissions/4 (pred) 499
pattern/1 (regtype) 500
do_options/1 (regtype) 500

do_str/3 (pred) . . . 500
do_str_without_nl/3 (pred) . . . 500
do_str_without_nl__popen/2 (pred) . . . 500
do_atmlist__popen/2 (pred) . . . 500
system_error_report/1 (udreexp) . . . 500
replace_characters/4 (udreexp) . . . 500
no_swapslash/3 (udreexp) . . . 500
cyg2win/3 (udreexp) . . . 501
winpath_c/3 (udreexp) . . . 501
winpath/3 (udreexp) . . . 501
winpath/2 (udreexp) . . . 501
using_windows/0 (udreexp) . . . 501
rename_file/2 (udreexp) . . . 501
delete_directory/1 (udreexp) . . . 501
delete_file/1 (udreexp) . . . 501
set_exec_mode/2 (udreexp) . . . 501
chmod/3 (udreexp) . . . 501
chmod/2 (udreexp) . . . 501
fmode/2 (udreexp) . . . 502
modif_time0/2 (udreexp) . . . 502
modif_time/2 (udreexp) . . . 502
file_properties/6 (udreexp) . . . 502
file_property/2 (udreexp) . . . 502
file_exists/2 (udreexp) . . . 502
file_exists/1 (udreexp) . . . 502
mktemp_in_tmp/2 (udreexp) . . . 502
mktemp/2 (udreexp) . . . 502
directory_files/2 (udreexp) . . . 502
wait/3 (udreexp) . . . 502
exec/8 (udreexp) . . . 502
exec/3 (udreexp) . . . 503
exec/4 (udreexp) . . . 503
popen_mode/1 (udreexp) . . . 503
popen/3 (udreexp) . . . 503
system/2 (udreexp) . . . 503
system/1 (udreexp) . . . 503
shell/2 (udreexp) . . . 503
shell/1 (udreexp) . . . 503
shell/0 (udreexp) . . . 503
cd/1 (udreexp) . . . 503
working_directory/2 (udreexp) . . . 503
make_dirpath/1 (udreexp) . . . 503
make_dirpath/2 (udreexp) . . . 504
make_directory/1 (udreexp) . . . 504
make_directory/2 (udreexp) . . . 504
umask/2 (udreexp) . . . 504
current_executable/1 (udreexp) . . . 504
current_host/1 (udreexp) . . . 504
get_address/2 (udreexp) . . . 504
get_tmp_dir/1 (udreexp) . . . 504
get_grnam/1 (udreexp) . . . 504
get_pwnam/1 (udreexp) . . . 504
get_gid/1 (udreexp) . . . 504
get_uid/1 (udreexp) . . . 504
get_pid/1 (udreexp) . . . 505
file_dir_name/3 (udreexp) . . . 505

extract_paths/2 (udreexp) 505
dir_path/2 (udreexp) 505
copy_file/3 (udreexp) 505
copy_file/2 (udreexp) 505
c_errno/1 (udreexp) 505
del_env/1 (udreexp) 505
set_env/2 (udreexp) 505
current_env/2 (udreexp) 505
setenvstr/2 (udreexp) 505
getenvstr/2 (udreexp) 505
datime_struct/1 (udreexp) 506
datime/9 (udreexp) 506
datime/1 (udreexp) 506
time/1 (udreexp) 506
pause/1 (udreexp) 506

PART VII - Ciao extensions 507

95 Pure Prolog package 509
95.1 Usage and interface (`pure_doc`) 510
95.2 Known bugs and planned improvements (`pure_doc`) 510

96 Multiple Argument Indexing 511
96.1 Usage and interface (`indexer_doc`) 511
96.2 Documentation on exports (`indexer_doc`) 511
hash_term/2 (pred) 511
96.3 Documentation on internals (`indexer_doc`) 512
index/1 (decl) 512
indexspecs/1 (regtype) 512
argspec/1 (regtype) 513

97 Higher-order 515
97.1 Usage and interface (`hiord_rt`) 515
97.2 Documentation on exports (`hiord_rt`) 515
call/1 (pred) 515
call/2 (pred) 515
SYSCALL/1 (pred) 516
$nodebug_call/1 (pred) 516
$meta_call/1 (pred) 516
97.3 Known bugs and planned improvements (`hiord_rt`) 516

98 Higher-order predicates 517
98.1 Usage and interface (`hiordlib`) 517
98.2 Documentation on exports (`hiordlib`) 517
map/3 (pred) 517
map/4 (pred) 518
foldl/4 (pred) 518
minimum/3 (pred) 518
split/4 (pred) 519

99 Terms with named arguments -records/feature terms ... 521
99.1 Usage and interface (argnames_doc) ... 521
99.2 Documentation on new declarations (argnames_doc) ... 521
(argnames)/1 (decl) ... 521
99.3 Documentation on exports (argnames_doc) ... 522
$~ /3 (pred) ... 522
99.4 Other information (argnames_doc) ... 522
99.4.1 Using argument names in a toy database ... 523
99.4.2 Complete code for the zebra example ... 523
99.5 Known bugs and planned improvements (argnames_doc) ... 524

100 Functional notation ... 525
100.1 Function applications ... 525
100.2 Predefined evaluable functors ... 525
100.3 Functional definitions ... 526
100.4 Quoting functors ... 526
100.5 Some scoping issues ... 527
100.6 Other functionality ... 527
100.7 Combining with higher order ... 527
100.8 Usage and interface (fsyntax_doc) ... 528
100.9 Other information (fsyntax_doc) ... 528
100.10 Some examples using functional syntax ... 528
100.11 Examples of combining with higher order ... 532
100.12 Some additional examples using functional syntax ... 533
100.13 Known bugs and planned improvements (fsyntax_doc) ... 535

101 global (library) ... 537
101.1 Usage and interface (global) ... 537
101.2 Documentation on exports (global) ... 537
set_global/2 (pred) ... 537
get_global/2 (pred) ... 537
push_global/2 (pred) ... 537
pop_global/2 (pred) ... 537
del_global/1 (pred) ... 537

102 Andorra execution ... 539
102.1 Usage and interface (andorra_doc) ... 539
102.2 Documentation on new declarations (andorra_doc) ... 539
determinate/2 (decl) ... 539
102.3 Documentation on exports (andorra_doc) ... 540
detcond/1 (regtype) ... 540
path/1 (regtype) ... 541
102.4 Other information (andorra_doc) ... 541

103 And-parallel execution ... 543
103.1 Usage and interface (andprolog_doc) ... 543

104 Low-level concurrency primitives for and-parallelism support 545

104.1 Usage and interface (apll) 546
104.2 Documentation on new declarations (apll) 547
(regtype)/1 (udreexp) 547
(regtype)/2 (decl) 547
104.3 Documentation on new modes (apll) 547
(+)/1 (modedef) 547
(@)/1 (modedef) 547
(-)/1 (modedef) 547
(?)/1 (modedef) 547
* /1 (modedef) 547
(+)/2 (modedef) 547
(@)/2 (modedef) 548
(-)/2 (modedef) 548
(?)/2 (modedef) 548
* /2 (modedef) 549
in/1 (modedef) 549
out/1 (modedef) 549
go/1 (modedef) 549
in/2 (modedef) 549
out/2 (modedef) 549
go/2 (modedef) 550
104.4 Documentation on exports (apll) 550
initial/0 (pred) 550
$start_thread/1 (pred) 550
$number_agents/1 (pred) 550
$push_goal/3 (pred) 551
$find_goal/3 (pred) 551
$goal_available/1 (pred) 551
$cancellation/1 (pred) 551
$retrieve_goal/2 (pred) 552
$goal_det/1 (pred) 552
$set_goal_det/1 (pred) 552
$set_goal_nondet/1 (pred) 552
$goal_not_executed/1 (pred) 552
$set_goal_not_executed/1 (pred) 553
$goal_rem_executing/1 (pred) 553
$set_goal_rem_executing/1 (pred) 553
$goal_finished/1 (pred) 553
$set_goal_finished/1 (pred) 554
$goal_tobacktrack/1 (pred) 554
$set_goal_tobacktrack/1 (pred) 554
$goal_toreexecute/1 (pred) 554
$set_goal_toreexecute/1 (pred) 554
$goal_failed/1 (pred) 555
$set_goal_failed/1 (pred) 555
$goal_cancelled/1 (pred) 555
$show_handler/1 (pred) 555
$set_goal_cancelled/1 (pred) 555
$send_event/1 (pred) 556
$read_event/1 (pred) 556
$save_init_execution/1 (pred) 556
$save_end_execution/1 (pred) 556
$more_solutions/1 (pred) 556
$move_execution_top/1 (pred) 557

$waiting/1 (pred) ... 557
$suspend/0 (pred) ... 557
$release/1 (pred) ... 557
$release_remote/1 (pred) ... 557
$release_some_suspended_thread/0 (pred) ... 558
$release_all_for_unwinding/0 (pred) ... 558
$enter_mutex/1 (pred) ... 558
$enter_mutex_self/0 (pred) ... 558
$enter_mutex_remote/1 (pred) ... 558
$exit_mutex/1 (pred) ... 559
$exit_mutex_self/0 (pred) ... 559
$exit_mutex_remote/1 (pred) ... 559
$clean_measures/0 (pred) ... 559
$print_measures/0 (pred) ... 559
$new_measure/0 (pred) ... 560
$not_measure/0 (pred) ... 560
$incr_num_local_backtr/0 (pred) ... 560
start_thread/1 (pred) ... 560
number_agents/1 (pred) ... 560
push_goal/3 (pred) ... 560
push_goal/1 (pred) ... 561
find_goal/2 (pred) ... 561
find_det_goal/2 (pred) ... 561
goal_available/1 (pred) ... 561
cancellation/1 (pred) ... 562
retrieve_goal/2 (pred) ... 562
goal_det/1 (pred) ... 562
set_goal_det/1 (pred) ... 562
set_goal_nondet/1 (pred) ... 562
goal_not_executed/1 (pred) ... 563
set_goal_not_executed/1 (pred) ... 563
goal_rem_executing/1 (pred) ... 563
set_goal_rem_executing/1 (pred) ... 563
goal_finished/1 (pred) ... 563
set_goal_finished/1 (pred) ... 563
goal_tobacktrack/1 (pred) ... 564
set_goal_tobacktrack/1 (pred) ... 564
goal_toreexecute/1 (pred) ... 564
set_goal_toreexecute/1 (pred) ... 564
goal_failed/1 (pred) ... 564
set_goal_failed/1 (pred) ... 565
show_handler/1 (pred) ... 565
goal_cancelled/1 (pred) ... 565
set_goal_cancelled/1 (pred) ... 565
send_event/1 (pred) ... 565
read_event/1 (pred) ... 566
save_init_execution/1 (pred) ... 566
save_end_execution/1 (pred) ... 566
more_solutions/1 (pred) ... 566
move_execution_top/1 (pred) ... 566
waiting/1 (pred) ... 566
suspend/0 (pred) ... 567
release/1 (pred) ... 567
release_remote/1 (pred) ... 567
release_some_suspended_thread/0 (pred) ... 567
release_all_for_unwinding/0 (pred) ... 567

enter_mutex/1 (pred) ... 567
enter_mutex_self/0 (pred) ... 568
enter_mutex_remote/1 (pred) ... 568
exit_mutex/1 (pred) ... 568
exit_mutex_self/0 (pred) ... 568
exit_mutex_remote/1 (pred) ... 568
clean_measures/0 (pred) ... 568
print_measures/0 (pred) ... 569
new_measure/0 (pred) ... 569
not_measure/0 (pred) ... 569
incr_num_local_backtr/0 (pred) ... 569

105 Call on determinate ... 571

105.1 Usage and interface (det_hook_doc) ... 571
105.2 Documentation on new modes (det_hook_doc) ... 571
(+)/1 (modedef) ... 571
(@)/1 (modedef) ... 571
(-)/1 (modedef) ... 571
(?)/1 (modedef) ... 571
* /1 (modedef) ... 572
(+)/2 (modedef) ... 572
(@)/2 (modedef) ... 572
(-)/2 (modedef) ... 572
(?)/2 (modedef) ... 572
* /2 (modedef) ... 572
105.3 Other information (det_hook_doc) ... 573
105.4 Known bugs and planned improvements (det_hook_doc) ... 573

106 Runtime predicates for call on determinate ... 575

106.1 Usage and interface (det_hook_rt) ... 575
106.2 Documentation on exports (det_hook_rt) ... 575
det_try/3 (pred) ... 575

107 Miscellaneous predicates ... 577

107.1 Usage and interface (odd) ... 577
107.2 Documentation on exports (odd) ... 577
setarg/3 (pred) ... 577
undo/1 (pred) ... 577

108 Delaying predicates (freeze) ... 579

108.1 Usage and interface (freeze) ... 579
108.2 Documentation on exports (freeze) ... 579
freeze/2 (pred) ... 579
frozen/2 (pred) ... 579
108.3 Known bugs and planned improvements (freeze) ... 579

109 Delaying predicates (when) ... 581

109.1 Usage and interface (when) ... 582
109.2 Documentation on exports (when) ... 582
when/2 (pred) ... 582
wakeup_exp/1 (regtype) ... 582
109.3 Known bugs and planned improvements (when) ... 583

110 Active modules (high-level distributed execution) ... 585
110.1 Active modules as agents ... 586
110.2 Usage and interface (actmods_doc) ... 586
110.3 Documentation on new declarations (actmods_doc) ... 587
use_active_module/2 (decl) ... 587
110.4 Other information (actmods_doc) ... 587
110.5 Active module name servers (webbased protocol) ... 587
110.6 Platforms (platformbased protocol) ... 588
110.7 Known bugs and planned improvements (actmods_doc) ... 588

111 Agents ... 589
111.1 Usage and interface (agent_doc) ... 589
111.2 Documentation on new declarations (agent_doc) ... 589
protocol/1 (decl) ... 589
111.3 Documentation on multifiles (agent_doc) ... 590
save_addr_actmod/1 (pred) ... 590
111.4 Documentation on internals (agent_doc) ... 590
module_address/2 (pred) ... 590
:: /2 (pred) ... 590
self/1 (pred) ... 590
111.5 Other information (agent_doc) ... 590
111.5.1 Platforms ... 590
111.6 Known bugs and planned improvements (agent_doc) ... 591

112 Breadth-first execution ... 593
112.1 Usage and interface (bf_doc) ... 594
112.2 Known bugs and planned improvements (bf_doc) ... 594

113 Iterative-deepening execution ... 595
113.1 Usage and interface (id_doc) ... 596

114 Constraint programming over rationals ... 597
114.1 Usage and interface (clpq_doc) ... 597
114.2 Other information (clpq_doc) ... 597
114.2.1 Some CLP(Q) examples ... 597
114.2.2 Meta-programming with CLP(Q) ... 598
114.3 Known bugs and planned improvements (clpq_doc) ... 599

115 Constraint programming over reals ... 601
115.1 Usage and interface (clpr_doc) ... 601
115.2 Other information (clpr_doc) ... 601
115.2.1 Some CLP(R) examples ... 601
115.2.2 Meta-programming with CLP(R) ... 603
115.3 Known bugs and planned improvements (clpr_doc) ... 603

116 Fuzzy Prolog 605

116.1 Usage and interface (`fuzzy_doc`) 605
116.2 Documentation on new declarations (`fuzzy_doc`) 606
aggr/1 (decl) 606
116.3 Documentation on exports (`fuzzy_doc`) 606
:# /2 (pred) 606
fuzzy_predicate/1 (pred) 606
fuzzy/1 (pred) 606
fnot/1 (pred) 607
:~ /2 (pred) 607
fuzzybody/1 (prop) 608
faggregator/1 (regtype) 608
=> /4 (pred) 608
116.4 Other information (`fuzzy_doc`) 609
116.5 Known bugs and planned improvements (`fuzzy_doc`) 609

117 Object Oriented Programming 611

117.1 Early examples 611
117.2 Recommendations on when to use objects 615
117.3 Limitations on object usage 615

118 Declaring classes and interfaces 617

118.1 Usage and interface (`class_doc`) 617
118.2 Documentation on new declarations (`class_doc`) 618
export/1 (decl) 618
public/1 (decl) 618
inheritable/1 (decl) 618
(data)/1 (decl) 618
(dynamic)/1 (decl) 619
(concurrent)/1 (decl) 619
inherit_class/1 (decl) 619
implements/1 (decl) 620
virtual/1 (decl) 620
118.3 Documentation on exports (`class_doc`) 621
inherited/1 (pred) 621
self/1 (pred) 621
constructor/0 (pred) 621
destructor/0 (pred) 622
118.4 Other information (`class_doc`) 622
118.4.1 Class and Interface error reporting at compile time 623
118.4.2 Class and Interface error reporting at run time 626
118.4.3 Normal Prolog module system interaction 626
118.5 Known bugs and planned improvements (`class_doc`) 627

119 Compile-time usage of objects 629

119.1 Usage and interface (`objects_doc`) 629
119.2 Documentation on new declarations (`objects_doc`) 629
use_class/1 (decl) 629
instance_of/2 (decl) 629
new/2 (decl) 630
119.3 Other information (`objects_doc`) 630
119.3.1 Error reporting at compile time (objects) 631
119.3.2 Error reporting at run time (objects) 632

120 Run time usage of objects 635
120.1 Usage and interface (`objects_rt`) 635
120.2 Documentation on exports (`objects_rt`) 635
new/2 (pred) 635
instance_of/2 (pred) 636
derived_from/2 (pred) 637
interface/2 (pred) 637
instance_codes/2 (pred) 637
destroy/1 (pred) 638
use_class/1 (pred) 638
constructor/1 (prop) 638
class_name/1 (prop) 638
interface_name/1 (prop) 639
instance_id/1 (prop) 639
class_source/1 (prop) 639
interface_source/1 (prop) 639
method_spec/1 (prop) 639
virtual_method_spec/1 (prop) 639
120.3 Known bugs and planned improvements (`objects_rt`) 639

121 Declaring abstract interfaces for classes 641
121.1 Usage and interface (`interface_doc`) 641

PART VIII - Interfaces to other languages and systems 643

122 Foreign Language Interface 645
122.1 Declaration of Types 645
122.2 Equivalence between Ciao Prolog and C types 645
122.3 Equivalence between Ciao Prolog and C modes 646
122.4 Custom access to Prolog from C 646
122.4.1 Term construction 647
122.4.2 Testing the Type of a Term 648
122.4.3 Term navigation 648
122.4.4 Testing for Equality and Performing Unification ... 649
122.4.5 Raising Exceptions 649
122.4.6 Creating and disposing of memory chunks 650
122.4.7 Calling Prolog from C 650
122.5 Examples 650
122.5.1 Mathematical functions 651
122.5.2 Addresses and C pointers 651
122.5.3 Lists of bytes and buffers 652
122.5.4 Lists of integers 653
122.5.5 Strings and atoms 654
122.5.6 Arbitrary Terms 655
122.5.7 Exceptions 657
122.5.8 Testing number types and using unbound length integers 657
122.6 Usage and interface (`foreign_interface_doc`) 659

123 Foreign Language Interface Properties 661

123.1 Usage and interface (`foreign_interface_properties`) 661
123.2 Documentation on exports (`foreign_interface_properties`) 661
int_list/1 (regtype) 661
double_list/1 (regtype) 661
byte_list/1 (regtype) 661
byte/1 (regtype) 661
null/1 (regtype) 661
address/1 (regtype) 662
any_term/1 (regtype) 662
foreign_low/1 (prop) 662
foreign_low/2 (prop) 662
size_of/3 (prop) 662
foreign/1 (prop) 662
foreign/2 (prop) 662
returns/2 (prop) 662
do_not_free/2 (prop) 663
needs_state/1 (prop) 663
ttr/3 (prop) 663
123.3 Documentation on internals (`foreign_interface_properties`) 663
use_foreign_source/1 (decl) 663
use_foreign_source/2 (decl) 663
use_foreign_library/1 (decl) 663
use_foreign_library/2 (decl) 663
extra_compiler_opts/1 (decl) 664
extra_compiler_opts/2 (decl) 664
use_compiler/1 (decl) 664
use_compiler/2 (decl) 664
extra_linker_opts/1 (decl) 664
extra_linker_opts/2 (decl) 665
use_linker/1 (decl) 665
use_linker/2 (decl) 665
foreign_inline/2 (decl) 665
123.4 Known bugs and planned improvements (`foreign_interface_properties`) 665

124 Utilities for on-demand compilation of foreign files 667

124.1 Usage and interface (`foreign_compilation`) 667
124.2 Documentation on exports (`foreign_compilation`) 667
compiler_and_opts/2 (pred) 667
linker_and_opts/2 (pred) 667

125 Foreign Language Interface Builder 669

125.1 Usage and interface (`build_foreign_interface`) 669
125.2 Documentation on exports (`build_foreign_interface`) . . . 669
build_foreign_interface/1 (pred) 669
rebuild_foreign_interface/1 (pred) 669
build_foreign_interface_explicit_decls/2 (pred) 670
rebuild_foreign_interface_explicit_decls/2 (pred) 670
build_foreign_interface_object/1 (pred) 670
rebuild_foreign_interface_object/1 (pred) 671
do_interface/1 (pred) 671

126 Interactive Menus 673
126.1 Usage and interface (menu_doc) 673
126.2 Documentation on multifiles (menu_doc) 673
menu_default/3 (pred) 673
menu_opt/6 (pred) 673
hook_menu_flag_values/3 (pred) 673
hook_menu_check_flag_value/3 (pred) 673
hook_menu_flag_help/3 (pred) 673
hook_menu_default_option/3 (pred) 674

127 menu_generator (library) 675
127.1 Usage and interface (menu_generator) 675
127.2 Documentation on exports (menu_generator) 675
menu/1 (pred) 675
menu/2 (pred) 675
menu/3 (pred) 675
menu/4 (pred) 675
get_menu_flag/3 (pred) 676
set_menu_flag/3 (pred) 676
space/1 (pred) 676
get_menu_configs/1 (pred) 676
save_menu_config/1 (pred) 676
remove_menu_config/1 (pred) 677
restore_menu_config/1 (pred) 677
show_menu_configs/0 (pred) 677
show_menu_config/1 (pred) 677
get_menu_options/2 (pred) 677
get_menu_flags/1 (pred) 677
restore_menu_flags_list/1 (pred) 678
get_menu_flags/2 (pred) 678
restore_menu_flags/2 (pred) 678
generate_js_menu/1 (pred) 678
eq/3 (pred) 681
neq/3 (pred) 681
uni_type/2 (pred) 681
vmember/2 (pred) 681
menu_flag_values/1 (regtype) 681
127.3 Documentation on multifiles (menu_generator) 681
$is_persistent/2 (pred) 681
persistent_dir/2 (pred) 681
persistent_dir/4 (pred) 682
menu_default/3 (pred) 682
menu_opt/6 (pred) 682
hook_menu_flag_values/3 (pred) 683
hook_menu_check_flag_value/3 (pred) 683
hook_menu_flag_help/3 (pred) 684
hook_menu_default_option/3 (pred) 684
127.4 Known bugs and planned improvements (menu_generator) 684

128 Interface to daVinci 685

128.1 Usage and interface (`davinci`) 685
128.2 Documentation on exports (`davinci`) 685
davinci/0 (pred) 685
topd/0 (pred) 685
davinci_get/1 (pred) 685
davinci_get_all/1 (pred) 685
davinci_put/1 (pred) 686
davinci_quit/0 (pred) 686
davinci_ugraph/1 (pred) 686
davinci_lgraph/1 (pred) 686
ugraph2term/2 (pred) 686
formatting/2 (pred) 686
128.3 Documentation on internals (`davinci`) 686
davinci_command/1 (prop) 686
ugraph/1 (prop) 687
lgraph/1 (prop) 687

129 The Tcl/Tk interface 689

129.1 Usage and interface (`tcltk`) 692
129.2 Documentation on exports (`tcltk`) 692
tcl_new/1 (pred) 692
tcl_eval/3 (pred) 692
tcl_delete/1 (pred) 693
tcl_event/3 (pred) 693
tclInterpreter/1 (regtype) 693
tclCommand/1 (regtype) 694
tk_event_loop/1 (pred) 694
tk_main_loop/1 (pred) 694
tk_new/2 (pred) 694
tk_next_event/2 (pred) 695

130 Low level interface library to Tcl/Tk 697

130.1 Usage and interface (`tcltk_low_level`) 697
130.2 Documentation on exports (`tcltk_low_level`) 697
new_interp/1 (pred) 697
new_interp/2 (pred) 697
new_interp_file/2 (pred) 698
tcltk/2 (pred) 698
tcltk_raw_code/2 (pred) 698
receive_result/2 (pred) 698
send_term/2 (pred) 699
receive_event/2 (pred) 699
receive_list/2 (pred) 699
receive_confirm/2 (pred) 699
delete/1 (pred) 700
130.3 Documentation on internals (`tcltk_low_level`) 700
core/1 (pred) 700

131 The PiLLoW Web programming library . . . 701

131.1 Installing PiLLoW 701
131.2 Usage and interface (`pillow_doc`) 701

132 HTML/XML/CGI programming 703

132.1 Usage and interface (`html`) 703
132.2 Documentation on exports (`html`) 703
output_html/1 (pred) 703
html2terms/2 (pred) 703
xml2terms/2 (pred) 704
html_template/3 (pred) 704
html_report_error/1 (pred) 706
get_form_input/1 (pred) 706
get_form_value/3 (pred) 706
form_empty_value/1 (pred) 706
form_default/3 (pred) 706
set_cookie/2 (pred) 707
get_cookies/1 (pred) 707
url_query/2 (pred) 707
url_query_amp/2 (pred) 707
url_query_values/2 (pred) 708
my_url/1 (pred) 708
url_info/2 (pred) 708
url_info_relative/3 (pred) 709
form_request_method/1 (pred) 710
icon_address/2 (pred) 710
html_protect/1 (pred) 710
http_lines/3 (pred) 710
132.3 Documentation on multifiles (`html`) 711
define_flag/3 (pred) 711
html_expansion/2 (pred) 711
132.4 Other information (`html`) 711

133 HTTP conectivity 713

133.1 Usage and interface (`http`) 713
133.2 Documentation on exports (`http`) 713
fetch_url/3 (pred) 713

134 PiLLoW types 715

134.1 Usage and interface (`pillow_types`) 715
134.2 Documentation on exports (`pillow_types`) 715
canonic_html_term/1 (regtype) 715
canonic_xml_term/1 (regtype) 716
html_term/1 (regtype) 717
form_dict/1 (regtype) 719
form_assignment/1 (regtype) 719
form_value/1 (regtype) 719
value_dict/1 (regtype) 720
url_term/1 (regtype) 720
http_request_param/1 (regtype) 720
http_response_param/1 (regtype) 720
http_date/1 (regtype) 721
weekday/1 (regtype) 721
month/1 (regtype) 721
hms_time/1 (regtype) 721

135 Persistent predicate database 723

135.1 Introduction to persistent predicates 723
135.2 Persistent predicates, files, and relational databases 723
135.3 Using file-based persistent predicates 724
135.4 Implementation Issues 724
135.5 Defining an initial database 725
135.6 Using persistent predicates from the top level 725
135.7 Usage and interface (`persdbrt`) 725
135.8 Documentation on exports (`persdbrt`) 726
passerta_fact/1 (pred) 726
passertz_fact/1 (pred) 726
pretract_fact/1 (pred) 726
pretractall_fact/1 (pred) 726
asserta_fact/1 (pred) 726
assertz_fact/1 (pred) 727
retract_fact/1 (pred) 727
retractall_fact/1 (pred) 727
initialize_db/0 (pred) 727
make_persistent/2 (pred) 728
update_files/0 (pred) 728
update_files/1 (pred) 728
create/2 (pred) 728
meta_predname/1 (regtype) 728
directoryname/1 (regtype) 728
135.9 Documentation on multifiles (`persdbrt`) 729
$is_persistent/2 (pred) 729
persistent_dir/2 (pred) 729
persistent_dir/4 (pred) 729
135.10 Documentation on internals (`persdbrt`) 729
persistent/2 (decl) 729
keyword/1 (pred) 730
135.11 Known bugs and planned improvements (`persdbrt`) 730

136 Using the persdb library 731

136.1 An example of persistent predicates (static version) 731
136.2 An example of persistent predicates (dynamic version) 731
136.3 A simple application / a persistent queue 731

137 Filed predicates 733

137.1 Usage and interface (`factsdb_doc`) 733
137.2 Documentation on multifiles (`factsdb_doc`) 733
$factsdb$cached_goal/3 (pred) 733
137.3 Known bugs and planned improvements (`factsdb_doc`) 733

138 Filed predicates (runtime) 735
138.1 Usage and interface (`factsdb_rt`) 735
138.2 Documentation on exports (`factsdb_rt`) 735
asserta_fact/1 (pred) 735
assertz_fact/1 (pred) 735
call/1 (pred) 736
current_fact/1 (pred) 736
retract_fact/1 (pred) 736
138.3 Documentation on multifiles (`factsdb_rt`) 736
$factsdb$cached_goal/3 (pred) 736
persistent_dir/2 (pred) 736
file_alias/2 (pred) 737
138.4 Documentation on internals (`factsdb_rt`) 737
facts/2 (decl) 737
keyword/1 (pred) 737

139 sqltypes (library) 739
139.1 Usage and interface (`sqltypes`) 739
139.2 Documentation on exports (`sqltypes`) 739
sqltype/1 (regtype) 739
accepted_type/2 (pred) 739
get_type/2 (pred) 739
type_compatible/2 (pred) 740
type_union/3 (pred) 740
sybasetype/1 (regtype) 740
sybase2sqltypes_list/2 (pred) 740
sybase2sqltype/2 (pred) 741
postgrestype/1 (regtype) 741
postgres2sqltypes_list/2 (pred) 741
postgres2sqltype/2 (pred) 741

140 persdbtr_sql (library) 743
140.1 Usage and interface (`persdbtr_sql`) 743
140.2 Documentation on exports (`persdbtr_sql`) 743
sql_persistent_tr/2 (pred) 743
sql_goal_tr/2 (pred) 743
dbId/2 (pred) 743

141 pl2sqlinsert (library) 745
141.1 Usage and interface (`pl2sqlinsert`) 745
141.2 Documentation on exports (`pl2sqlinsert`) 745
pl2sqlInsert/2 (pred) 745
141.3 Documentation on multifiles (`pl2sqlinsert`) 745
sql__relation/3 (pred) 745
sql__attribute/4 (pred) 745

142 Prolog/Java Bidirectional Interface 747
142.1 Distributed Programming Model 747

143 Prolog to Java interface 749
143.1 Prolog to Java Interface Structure 749
143.1.1 Prolog side of the Java interface 749
143.1.2 Java side 749
143.2 Java event handling from Prolog 750
143.3 Java exception handling from Prolog 752
143.4 Usage and interface (`javart`) 752
143.5 Documentation on exports (`javart`) 752
java_start/0 (pred) 752
java_start/1 (pred) 752
java_start/2 (pred) 753
java_stop/0 (pred) 753
java_connect/2 (pred) 753
java_disconnect/0 (pred) 753
machine_name/1 (regtype) 753
java_constructor/1 (regtype) 754
java_object/1 (regtype) 754
java_event/1 (regtype) 754
prolog_goal/1 (regtype) 754
java_field/1 (regtype) 754
java_use_module/1 (pred) 754
java_create_object/2 (pred) 754
java_delete_object/1 (pred) 755
java_invoke_method/2 (pred) 755
java_method/1 (regtype) 755
java_get_value/2 (pred) 756
java_set_value/2 (pred) 756
java_add_listener/3 (pred) 756
java_remove_listener/3 (pred) 757

144 Java to Prolog interface 759
144.1 Usage and interface (`jtopl`) 759
144.2 Documentation on exports (`jtopl`) 759
prolog_server/0 (pred) 759
prolog_server/1 (pred) 760
prolog_server/2 (pred) 760
shell_s/0 (pred) 760
query_solutions/2 (pred) 760
query_requests/2 (pred) 760
running_queries/2 (pred) 760

145 Low-level Prolog to Java socket connection 763
145.1 Usage and interface (javasock) 763
145.2 Documentation on exports (javasock) 763
bind_socket_interface/1 (pred) 763
start_socket_interface/2 (pred) 763
stop_socket_interface/0 (pred) 764
join_socket_interface/0 (pred) 764
java_query/2 (pred) 764
java_response/2 (pred) 764
prolog_query/2 (pred) 764
prolog_response/2 (pred) 765
is_connected_to_java/0 (pred) 765
java_debug/1 (pred) 765
java_debug_redo/1 (pred) 765
start_threads/0 (pred) 765

146 Calling emacs from Prolog 767
146.1 Usage and interface (emacs) 768
146.2 Documentation on exports (emacs) 768
emacs_edit/1 (pred) 768
emacs_edit_nowait/1 (pred) 768
emacs_eval/1 (pred) 768
emacs_eval_nowait/1 (pred) 768
elisp_string/1 (regtype) 769

147 linda (library) 771
147.1 Usage and interface (linda) 771
147.2 Documentation on exports (linda) 771
linda_client/1 (pred) 771
close_client/0 (pred) 771
in/1 (pred) 771
in/2 (pred) 771
in_noblock/1 (pred) 771
out/1 (pred) 771
rd/1 (pred) 772
rd/2 (pred) 772
rd_noblock/1 (pred) 772
rd_findall/3 (pred) 772
linda_timeout/2 (pred) 772
halt_server/0 (pred) 772
open_client/2 (pred) 772
in_stream/2 (pred) 772
out_stream/2 (pred) 772

PART IX - Abstract data types 773

148 Extendable arrays with logarithmic access time 775

148.1 Usage and interface (`arrays`) 775
148.2 Documentation on exports (`arrays`) 775
new_array/1 (pred) 775
is_array/1 (pred) 775
aref/3 (pred) 775
arefa/3 (pred) 775
arefl/3 (pred) 776
aset/4 (pred) 776
array_to_list/2 (pred) 776

149 Association between key and value 777

149.1 Usage and interface (`assoc`) 777
149.2 Documentation on exports (`assoc`) 777
empty_assoc/1 (pred) 777
assoc_to_list/2 (pred) 777
is_assoc/1 (pred) 778
min_assoc/3 (pred) 778
max_assoc/3 (pred) 778
gen_assoc/3 (pred) 778
get_assoc/3 (pred) 779
get_assoc/5 (pred) 779
get_next_assoc/4 (pred) 780
get_prev_assoc/4 (pred) 780
list_to_assoc/2 (pred) 780
ord_list_to_assoc/2 (pred) 781
map_assoc/2 (pred) 781
map_assoc/3 (pred) 781
map/3 (pred) 781
foldl/4 (pred) 782
put_assoc/4 (pred) 782
put_assoc/5 (pred) 782
add_assoc/4 (pred) 783
update_assoc/5 (pred) 783
del_assoc/4 (pred) 783
del_min_assoc/4 (pred) 784
del_max_assoc/4 (pred) 784

150 counters (library) 785

150.1 Usage and interface (`counters`) 785
150.2 Documentation on exports (`counters`) 785
setcounter/2 (pred) 785
getcounter/2 (pred) 785
inccounter/2 (pred) 785

151 Identity lists 787
151.1 Usage and interface (idlists) 787
151.2 Documentation on exports (idlists) 787
member_0/2 (pred) 787
memberchk/2 (pred) 787
list_insert/2 (pred) 787
add_after/4 (pred) 787
add_before/4 (pred) 788
delete/3 (pred) 788
subtract/3 (pred) 788
union_idlists/3 (pred) 788

152 Lists of numbers 789
152.1 Usage and interface (numlists) 789
152.2 Documentation on exports (numlists) 789
get_primes/2 (pred) 789
intlist/1 (regtype) 789
numlist/1 (regtype) 789
sum_list/2 (pred) 789
sum_list/3 (pred) 790
sum_list_of_lists/2 (pred) 790
sum_list_of_lists/3 (pred) 790

153 Pattern (regular expression) matching -deprecated version 791
153.1 Usage and interface (patterns) 791
153.2 Documentation on exports (patterns) 791
match_pattern/2 (pred) 791
match_pattern/3 (pred) 791
case_insensitive_match/2 (pred) 791
letter_match/2 (pred) 792
pattern/1 (regtype) 792
match_pattern_pred/2 (pred) 792

154 Graphs 793
154.1 Usage and interface (graphs) 793
154.2 Documentation on exports (graphs) 793
dgraph/1 (regtype) 793
dlgraph/1 (regtype) 793
dgraph_to_ugraph/2 (pred) 793
dlgraph_to_lgraph/2 (pred) 794
edges_to_ugraph/2 (pred) 794
edges_to_lgraph/2 (pred) 794
154.3 Documentation on internals (graphs) 795
pair/1 (regtype) 795
triple/1 (regtype) 795

155 Unweighted graph-processing utilities 797
155.1 Usage and interface (`ugraphs`) 797
155.2 Documentation on exports (`ugraphs`) 797
vertices_edges_to_ugraph/3 (pred) 797
neighbors/3 (pred) 797
edges/2 (pred) 797
del_edges/3 (pred) 798
add_edges/3 (pred) 798
vertices/2 (pred) 798
del_vertices/3 (pred) 798
add_vertices/3 (pred) 798
transpose/2 (pred) 799
rooted_subgraph/3 (pred) 799
point_to/3 (pred) 799
ugraph/1 (regtype) 799

156 wgraphs (library) 801
156.1 Usage and interface (`wgraphs`) 801
156.2 Documentation on exports (`wgraphs`) 801
vertices_edges_to_wgraph/3 (pred) 801

157 Labeled graph-processing utilities 803
157.1 Usage and interface (`lgraphs`) 803
157.2 Documentation on exports (`lgraphs`) 803
lgraph/2 (regtype) 803
vertices_edges_to_lgraph/3 (pred) 803

158 queues (library) 805
158.1 Usage and interface (`queues`) 805
158.2 Documentation on exports (`queues`) 805
q_empty/1 (pred) 805
q_insert/3 (pred) 805
q_member/2 (pred) 805
q_delete/3 (pred) 805

159 Random numbers 807
159.1 Usage and interface (`random`) 807
159.2 Documentation on exports (`random`) 807
random/1 (pred) 807
random/3 (pred) 807
srandom/1 (pred) 808

160 Set Operations 809
160.1 Usage and interface (sets) 809
160.2 Documentation on exports (sets) 809
insert/3 (pred) 809
ord_delete/3 (pred) 809
ord_member/2 (pred) 809
ord_test_member/3 (pred) 810
ord_subtract/3 (pred) 810
ord_intersection/3 (pred) 810
ord_intersection_diff/4 (pred) 810
ord_intersect/2 (pred) 810
ord_subset/2 (pred) 811
ord_subset_diff/3 (pred) 811
ord_union/3 (pred) 811
ord_union_diff/4 (pred) 811
ord_union_symdiff/4 (pred) 812
ord_union_change/3 (pred) 812
merge/3 (pred) 812
ord_disjoint/2 (pred) 812
setproduct/3 (pred) 812

161 Variable name dictionaries 813
161.1 Usage and interface (vndict) 813
161.2 Documentation on exports (vndict) 813
null_dict/1 (regtype) 813
create_dict/2 (pred) 813
create_pretty_dict/2 (pred) 813
complete_dict/3 (pred) 814
complete_vars_dict/3 (pred) 814
prune_dict/3 (pred) 814
sort_dict/2 (pred) 814
dict2varnamesl/2 (pred) 814
varnamesl2dict/2 (pred) 815
find_name/4 (pred) 815
prettyvars/2 (pred) 815
rename/2 (pred) 815
varnamedict/1 (regtype) 815
vars_names_dict/3 (pred) 815

PART X - Miscellaneous standalone utilities 817

162 A Program to Help Cleaning your Directories 819
162.1 Usage (cleandirs) 819
162.2 Known bugs and planned improvements (cleandirs) 819

163 Printing the declarations and code in a file 821
163.1 Usage (fileinfo) 821
163.2 More detailed explanation of options (fileinfo) 821

164 Printing the contents of a bytecode file 823
164.1 Usage (viewpo) 823

165 callgraph (library) 825
165.1 Usage and interface (`callgraph`) 825
165.2 Documentation on exports (`callgraph`) 825
call_graph/2 (pred) 825
reachability/4 (pred) 825

166 Gathering the dependent files for a file 827
166.1 Usage (get_deps) 827

167 Finding differences between two Prolog files 829
167.1 Usage (pldiff) 829
167.2 Known bugs and planned improvements (`pldiff`) 829

168 The Ciao lpmake scripting facility 831
168.1 General operation 831
168.2 Format of the Configuration File 831
168.3 lpmake usage 832
168.4 Acknowledgments (`lpmake`) 833
168.5 Known bugs and planned improvements (`lpmake`) 833

169 Find out which architecture we are running on 835
169.1 Usage (ciao_get_arch) 835
169.2 More details (ciao_get_arch) 835

170 Print out WAM code 837
170.1 Usage (compiler_output) 837

171 Compile Ciao in an arbitrary subdirectory 839
171.1 Usage (auto_compile_ciao) 839
171.2 More details (auto_compile_ciao) 839

172 Gathering all modules in a directory 841
172.1 Usage (collect_modules) 841
172.2 More details (collect_modules) 841

PART XI - Contributed libraries 843

173 Block Declarations 845
173.1 Usage and interface (`block_doc`) 845
173.2 Documentation on new declarations (`block_doc`) 845
block/1 (decl) 845

174 A Chart Library 847
174.1 Bar charts 847
174.2 Line graphs 849
174.3 Scatter graphs 849
174.4 Tables 850
174.5 Overview of widgets 851
174.6 Usage and interface (`chartlib`) 852
174.7 Documentation on exports (`chartlib`) 852
barchart1/7 (udreexp) 852
barchart1/9 (udreexp) 852
percentbarchart1/7 (udreexp) 852
barchart2/7 (udreexp) 852
barchart2/11 (udreexp) 852
percentbarchart2/7 (udreexp) 852
barchart3/7 (udreexp) 852
barchart3/9 (udreexp) 852
percentbarchart3/7 (udreexp) 853
barchart4/7 (udreexp) 853
barchart4/11 (udreexp) 853
percentbarchart4/7 (udreexp) 853
multibarchart/8 (udreexp) 853
multibarchart/10 (udreexp) 853
tablewidget1/4 (udreexp) 853
tablewidget1/5 (udreexp) 853
tablewidget2/4 (udreexp) 853
tablewidget2/5 (udreexp) 853
tablewidget3/4 (udreexp) 853
tablewidget3/5 (udreexp) 853
tablewidget4/4 (udreexp) 854
tablewidget4/5 (udreexp) 854
graph_b1/9 (udreexp) 854
graph_b1/13 (udreexp) 854
graph_w1/9 (udreexp) 854
graph_w1/13 (udreexp) 854
scattergraph_b1/8 (udreexp) 854
scattergraph_b1/12 (udreexp) 854
scattergraph_w1/8 (udreexp) 854
scattergraph_w1/12 (udreexp) 854
graph_b2/9 (udreexp) 854
graph_b2/13 (udreexp) 854
graph_w2/9 (udreexp) 855
graph_w2/13 (udreexp) 855
scattergraph_b2/8 (udreexp) 855
scattergraph_b2/12 (udreexp) 855
scattergraph_w2/8 (udreexp) 855
scattergraph_w2/12 (udreexp) 855
chartlib_text_error_protect/1 (udreexp) 855
chartlib_visual_error_protect/1 (udreexp) 855
174.8 Known bugs and planned improvements (`chartlib`) 855

175 Low level Interface between Prolog and blt 857
175.1 Usage and interface (bltclass) 857
175.2 Documentation on exports (bltclass) 857
new_interp/1 (pred) 857
tcltk_raw_code/2 (pred) 857
bltwish_interp/1 (regtype) 857
interp_file/2 (pred) 858

176 Error Handler for Chartlib 859
176.1 Usage and interface (chartlib_errhandle) 859
176.2 Documentation on exports (chartlib_errhandle) 859
chartlib_text_error_protect/1 (pred) 859
chartlib_visual_error_protect/1 (pred) 859
176.3 Documentation on internals (chartlib_errhandle) 860
handler_type/1 (regtype) 860
error_message/2 (pred) 860
error_file/2 (pred) 860

177 Color and Pattern Library 861
177.1 Usage and interface (color_pattern) 861
177.2 Documentation on exports (color_pattern) 861
color/1 (regtype) 861
color/2 (pred) 862
pattern/1 (regtype) 863
pattern/2 (pred) 863
random_color/1 (pred) 863
random_lightcolor/1 (pred) 863
random_darkcolor/1 (pred) 864
random_pattern/1 (pred) 864

178 Barchart widgets - 1 865
178.1 Usage and interface (genbar1) 865
178.2 Documentation on exports (genbar1) 865
barchart1/7 (pred) 865
barchart1/9 (pred) 866
percentbarchart1/7 (pred) 867
yelement/1 (regtype) 867
axis_limit/1 (regtype) 868
header/1 (regtype) 869
title/1 (regtype) 869
footer/1 (regtype) 869
178.3 Documentation on internals (genbar1) 869
xbarelement1/1 (regtype) 869
178.4 Known bugs and planned improvements (genbar1) 870

179 Barchart widgets - 2 871
179.1 Usage and interface (genbar2) 871
179.2 Documentation on exports (genbar2) 871
barchart2/7 (pred) 871
barchart2/11 (pred) 872
percentbarchart2/7 (pred) 873
xbarelement2/1 (regtype) 873

180 Depict barchart widgets - 3 875
180.1 Usage and interface (genbar3) 875
180.2 Documentation on exports (genbar3) 875
barchart3/7 (pred) 875
barchart3/9 (pred) 876
percentbarchart3/7 (pred) 876
180.3 Documentation on internals (genbar3) 877
xbarelement3/1 (regtype) 877

181 Depict barchart widgets - 4 879
181.1 Usage and interface (genbar4) 879
181.2 Documentation on exports (genbar4) 879
barchart4/7 (pred) 879
barchart4/11 (pred) 880
percentbarchart4/7 (pred) 880
181.3 Documentation on internals (genbar4) 881
xbarelement4/1 (regtype) 881

182 Depic line graph 883
182.1 Usage and interface (gengraph1) 884
182.2 Documentation on exports (gengraph1) 884
graph_b1/9 (pred) 884
graph_b1/13 (pred) 885
graph_w1/9 (pred) 885
graph_w1/13 (pred) 886
scattergraph_b1/8 (pred) 887
scattergraph_b1/12 (pred) 887
scattergraph_w1/8 (pred) 888
scattergraph_w1/12 (pred) 889
vector/1 (regtype) 889
smooth/1 (regtype) 890
attributes/1 (regtype) 890
symbol/1 (regtype) 890
size/1 (regtype) 891

183 Line graph widgets 893
183.1 Usage and interface (gengraph2) 893
183.2 Documentation on exports (gengraph2) 893
graph_b2/9 (pred) 893
graph_b2/13 (pred) 894
graph_w2/9 (pred) 895
graph_w2/13 (pred) 896
scattergraph_b2/8 (pred) 896
scattergraph_b2/12 (pred) 897
scattergraph_w2/8 (pred) 898
scattergraph_w2/12 (pred) 898

184 Multi barchart widgets 901
184.1 Usage and interface (genmultibar) 901
184.2 Documentation on exports (genmultibar) 902
multibarchart/8 (pred) 902
multibarchart/10 (pred) 902
184.3 Documentation on internals (genmultibar) 903
multibar_attribute/1 (regtype) 903
xelement/1 (regtype) 904

185 table_widget1 (library) 905
185.1 Usage and interface (table_widget1) 905
185.2 Documentation on exports (table_widget1) 905
tablewidget1/4 (pred) 905
tablewidget1/5 (pred) 905
table/1 (regtype) 906
image/1 (regtype) 906
185.3 Documentation on internals (table_widget1) 906
row/1 (regtype) 906
row/1 (regtype) 906
cell_value/1 (regtype) 907

186 table_widget2 (library) 909
186.1 Usage and interface (table_widget2) 909
186.2 Documentation on exports (table_widget2) 909
tablewidget2/4 (pred) 909
tablewidget2/5 (pred) 909

187 table_widget3 (library) 911
187.1 Usage and interface (table_widget3) 911
187.2 Documentation on exports (table_widget3) 911
tablewidget3/4 (pred) 911
tablewidget3/5 (pred) 911

188 table_widget4 (library) 913
188.1 Usage and interface (table_widget4) 913
188.2 Documentation on exports (table_widget4) 913
tablewidget4/4 (pred) 913
tablewidget4/5 (pred) 913

189 test_format (library) 915
189.1 Usage and interface (test_format) 915
189.2 Documentation on exports (test_format) 915
equalnumber/3 (pred) 915
not_empty/4 (pred) 915
not_empty/3 (pred) 916
check_sublist/4 (pred) 916
valid_format/4 (pred) 916
vectors_format/4 (pred) 916
valid_vectors/4 (pred) 917
valid_attributes/2 (pred) 917
valid_table/2 (pred) 917

190 Doubly linked lists 919
190.1 Usage and interface (ddlist) 919
190.2 Documentation on exports (ddlist) 919
null_ddlist/1 (pred) 919
create_from_list/2 (pred) 919
to_list/2 (pred) 920
next/2 (pred) 920
prev/2 (pred) 920
insert/3 (pred) 920
insert_top/3 (pred) 920
insert_after/3 (pred) 921
insert_begin/3 (pred) 921
insert_end/3 (pred) 921
delete/2 (pred) 921
delete_top/2 (pred) 921
delete_after/2 (pred) 921
remove_all_elements/3 (pred) 922
top/2 (pred) 922
rewind/2 (pred) 922
forward/2 (pred) 922
length/2 (pred) 922
length_next/2 (pred) 923
length_prev/2 (pred) 923
ddlist/1 (regtype) 923
ddlist_member/2 (pred) 923
190.3 Other information (ddlist) 923
190.3.1 Using insert_after 924
190.3.2 More Complex example 924

Ciao DHT Implementation 927

191 Top-level user interface to DHT 929
191.1 Usage and interface (dht_client) 929
191.2 Documentation on exports (dht_client) 929
dht_connect/2 (pred) 929
dht_connect/3 (pred) 929
dht_disconnect/1 (pred) 930
dht_consult_b/4 (pred) 930
dht_consult_nb/4 (pred) 931
dht_extract_b/4 (pred) 931
dht_extract_nb/4 (pred) 931
dht_store/4 (pred) 932
dht_hash/3 (pred) 932

192 Top-level interface to a DHT server 935
192.1 Usage and interface (dht_server) 935
192.2 Documentation on exports (dht_server) 935
dht_server/1 (pred) 935
dht_prolog/1 (pred) 935

193 Server to client communication module 937
193.1 Usage and interface (dht_s2c) 937
193.2 Documentation on exports (dht_s2c) 937
dht_s2c_main/0 (pred) 937

194 Server to server communication module ... 939
194.1 Usage and interface (dht_s2s) ... 939
194.2 Documentation on exports (dht_s2s) ... 939
dht_s2s_main/0 (pred) ... 939

195 DHT-related logics ... 941
195.1 Usage and interface (dht_logic) ... 941
195.2 Documentation on exports (dht_logic) ... 941
dht_init/1 (pred) ... 941
dht_finger/2 (pred) ... 941
dht_successor/1 (pred) ... 942
dht_check_predecessor/1 (pred) ... 942
dht_closest_preceding_finger/2 (pred) ... 942
dht_find_predecessor/2 (pred) ... 943
dht_find_successor/2 (pred) ... 944
dht_join/1 (pred) ... 944
dht_notify/1 (pred) ... 944
dht_stabilize/0 (pred) ... 945
dht_fix_fingers/0 (pred) ... 945
dht_id_by_node/2 (pred) ... 946
dht_find_and_consult_b/2 (pred) ... 946
dht_consult_server_b/3 (pred) ... 946
dht_find_and_consult_nb/2 (pred) ... 947
dht_consult_server_nb/3 (pred) ... 948
dht_find_and_extract_b/2 (pred) ... 948
dht_extract_from_server_b/3 (pred) ... 949
dht_find_and_extract_nb/2 (pred) ... 950
dht_extract_from_server_nb/3 (pred) ... 950
dht_find_and_store/2 (pred) ... 951
dht_store_to_server/4 (pred) ... 951

196 Finger table and routing information ... 953
196.1 Usage and interface (dht_routing) ... 953
196.2 Documentation on exports (dht_routing) ... 953
dht_finger_table/2 (pred) ... 953
dht_finger_start/2 (pred) ... 954
dht_update_finger/2 (pred) ... 955
dht_set_finger/4 (pred) ... 955
dht_predecessor/1 (pred) ... 956
dht_set_predecessor/1 (pred) ... 956
dht_reset_predecessor/0 (pred) ... 956

197 Various wrappers for DHT logics module .. 957
197.1 Usage and interface (dht_logic_misc) ... 957
197.2 Documentation on exports (dht_logic_misc) ... 957
hash_size/1 (pred) ... 957
highest_hash_number/1 (pred) ... 957
consistent_hash/2 (pred) ... 958
next_on_circle/2 (pred) ... 958
not_in_circle_oc/3 (pred) ... 958
in_circle_oo/3 (pred) ... 959
in_circle_oc/3 (pred) ... 959

198 Remote predicate calling utilities 961
198.1 Usage and interface (dht_rpr) 961
198.2 Documentation on exports (dht_rpr) 961
dht_rpr_register_node/1 (pred) 961
dht_rpr_register_node/2 (pred) 961
dht_rpr_node_by_id/2 (pred) 962
dht_rpr_id_by_node/2 (pred) 962
dht_rpr_node_id/1 (regtype) 963
dht_rpr_compose_id/3 (pred) 964
dht_rpr_clear_by_node/1 (pred) 964
dht_rpr_node/1 (pred) 965
dht_rpr_call/2 (pred) 965
dht_rpr_call/3 (pred) 968
node_id/2 (pred) 969

199 Underlying data-storage module 971
199.1 Usage and interface (dht_storage) 971
199.2 Documentation on exports (dht_storage) 971
dht_store/3 (pred) 971
dht_extract_b/2 (pred) 971
dht_extract_nb/2 (pred) 972
dht_consult_b/2 (pred) 972
dht_consult_nb/2 (pred) 973
dht_key_hash/2 (pred) 973

200 Configuration module 975
200.1 Usage and interface (dht_config) 975
200.2 Documentation on exports (dht_config) 975
hash_power/1 (pred) 975
dht_set_hash_power/1 (pred) 975
dht_s2c_port/1 (pred) 975
dht_set_s2c_port/1 (pred) 976
dht_s2c_threads/1 (pred) 976
dht_set_s2c_threads/1 (pred) 976
dht_s2s_port/1 (pred) 976
dht_set_s2s_port/1 (pred) 976
dht_s2s_threads/1 (pred) 976
dht_set_s2s_threads/1 (pred) 976
dht_join_host/1 (pred) 976
dht_set_join_host/1 (pred) 977
dht_server_id/1 (pred) 977
dht_set_server_id/1 (pred) 977
dht_server_host/1 (pred) 977
dht_set_server_host/1 (pred) 977

201 Tiny module with miscellaneous functions .. 979
201.1 Usage and interface (dht_misc) 979
201.2 Documentation on exports (dht_misc) 979
write_pr/2 (pred) 979
read_pr/2 (pred) 979

202 Constraint programming over finite domains 981
202.1 Usage and interface (fd_doc) 982
202.2 Documentation on exports (fd_doc) 982
fd_item/1 (regtype) 982
fd_range/1 (regtype) 982
fd_subrange/1 (regtype) 982
fd_store/1 (regtype) 983
fd_store_entity/1 (regtype) 983
labeling/1 (pred) 983
pitm/2 (pred) 983
choose_var/3 (pred) 983
choose_free_var/2 (pred) 984
choose_var_nd/2 (pred) 984
choose_value/2 (pred) 984
retrieve_range/2 (pred) 984
retrieve_store/2 (pred) 984
glb/2 (pred) 985
lub/2 (pred) 985
bounds/3 (pred) 985
retrieve_list_of_values/2 (pred) 986

203 Dot generator 987
203.1 Usage and interface (gendot) 987
203.2 Documentation on exports (gendot) 987
gendot/3 (pred) 987

204 Printing graphs using gnuplot as auxiliary tool 989
204.1 Usage and interface (gnuplot) 989
204.2 Documentation on exports (gnuplot) 989
get_general_options/1 (pred) 989
set_general_options/1 (pred) 989
generate_plot/2 (pred) 990
generate_plot/3 (pred) 990

205 Lazy evaluation 993
205.1 Usage and interface (lazy_doc) 995
205.2 Other information (lazy_doc) 995

206 Programming MYCIN rules 997
206.1 Usage and interface (mycin_doc) 997
206.2 Documentation on new declarations (mycin_doc) 997
export/1 (decl) 997
206.3 Known bugs and planned improvements (mycin_doc) 997

207 The Ciao Profiler 999
207.1 Usage and interface (profiler_doc) 999

208 ProVRML - a Prolog interface for VRML 1001
208.1 Usage and interface (provrml) 1001
208.2 Documentation on exports (provrml) 1001
vrml_web_to_terms/2 (pred) 1001
vrml_file_to_terms/2 (pred) 1001
vrml_web_to_terms_file/2 (pred) 1002
vrml_file_to_terms_file/2 (pred) 1002
terms_file_to_vrml/2 (pred) 1002
terms_file_to_vrml_file/2 (pred) 1003
terms_to_vrml_file/2 (pred) 1003
terms_to_vrml/2 (pred) 1003
vrml_to_terms/2 (pred) 1003
vrml_in_out/2 (pred) 1004
vrml_http_access/2 (pred) 1004
208.3 Documentation on internals (provrml) 1004
read_page/2 (pred) 1004

209 boundary (library) 1005
209.1 Usage and interface (boundary) 1005
209.2 Documentation on exports (boundary) 1005
boundary_check/3 (pred) 1005
boundary_rotation_first/2 (pred) 1005
boundary_rotation_last/2 (pred) 1006
reserved_words/1 (pred) 1006
children_nodes/1 (pred) 1006

210 dictionary (library) 1007
210.1 Usage and interface (dictionary) 1007
210.2 Documentation on exports (dictionary) 1007
dictionary/6 (pred) 1007

211 dictionary_tree (library) 1009
211.1 Usage and interface (dictionary_tree) 1009
211.2 Documentation on exports (dictionary_tree) 1009
create_dictionaries/1 (pred) 1009
is_dictionaries/1 (pred) 1009
get_definition_dictionary/2 (pred) 1009
get_prototype_dictionary/2 (pred) 1010
dictionary_insert/5 (pred) 1010
dictionary_lookup/5 (pred) 1010
merge_tree/2 (pred) 1011

212 provrmlerror (library) 1013
212.1 Usage and interface (provrmlerror) 1013
212.2 Documentation on exports (provrmlerror) 1013
error_vrml/1 (pred) 1013
output_error/1 (pred) 1013

213 field_type (library) 1015
213.1 Usage and interface (field_type) 1015
213.2 Documentation on exports (field_type) 1015
fieldType/1 (pred) 1015

214 field_value (library) 1017
214.1 Usage and interface (field_value) 1017
214.2 Documentation on exports (field_value) 1017
fieldValue/6 (pred) 1017
mfstringValue/5 (pred) 1017
parse/1 (prop) 1018

215 field_value_check (library) 1019
215.1 Usage and interface (field_value_check) 1019
215.2 Documentation on exports (field_value_check) 1019
fieldValue_check/8 (pred) 1019
mfstringValue/7 (pred) 1020

216 generator (library) 1021
216.1 Usage and interface (generator) 1021
216.2 Documentation on exports (generator) 1021
generator/2 (pred) 1021
nodeDeclaration/4 (pred) 1021

217 generator_util (library) 1023
217.1 Usage and interface (generator_util) 1023
217.2 Documentation on exports (generator_util) 1023
reading/4 (pred) 1023
reading/5 (pred) 1025
reading/6 (pred) 1029
open_node/6 (pred) 1029
close_node/5 (pred) 1029
close_nodeGut/4 (pred) 1030
open_PROTO/4 (pred) 1030
close_PROTO/6 (pred) 1030
open_EXTERNPROTO/5 (pred) 1031
close_EXTERNPROTO/6 (pred) 1031
open_DEF/5 (pred) 1031
close_DEF/5 (pred) 1032
open_Script/5 (pred) 1032
close_Script/5 (pred) 1033
decompose_field/3 (pred) 1033
indentation_list/2 (pred) 1033
start_vrmlScene/4 (pred) 1033
remove_comments/4 (pred) 1034
217.3 Known bugs and planned improvements (generator_util) 1034

218 internal_types (library) 1035
218.1 Usage and interface (internal_types) 1035
218.2 Documentation on exports (internal_types) 1035
bound/1 (regtype) 1035
bound_double/1 (regtype) 1035
dictionary/1 (regtype) 1035
environment/1 (regtype) 1036
parse/1 (regtype) 1036
tree/1 (regtype) 1036
whitespace/1 (regtype) 1036

219 provrml_io (library) 1039

219.1 Usage and interface (`provrml_io`) 1039
219.2 Documentation on exports (`provrml_io`) 1039
out/1 (pred) 1039
out/3 (pred) 1039
convert_atoms_to_string/2 (pred) 1039
read_terms_file/2 (pred) 1040
write_terms_file/2 (pred) 1040
read_vrml_file/2 (pred) 1040
write_vrml_file/2 (pred) 1040

220 lookup (library) 1041

220.1 Usage and interface (`lookup`) 1041
220.2 Documentation on exports (`lookup`) 1041
create_proto_element/3 (pred) 1041
get_prototype_interface/2 (pred) 1041
get_prototype_definition/2 (pred) 1042
lookup_check_node/4 (pred) 1042
lookup_check_field/6 (pred) 1042
lookup_check_interface_fieldValue/8 (pred) 1043
lookup_field/4 (pred) 1043
lookup_route/5 (pred) 1044
lookup_fieldTypeId/1 (pred) 1044
lookup_get_fieldType/4 (pred) 1044
lookup_field_access/4 (pred) 1044
lookup_set_def/3 (pred) 1045
lookup_set_prototype/4 (pred) 1045
lookup_set_extern_prototype/4 (pred) 1046

221 provrml_parser (library) 1047

221.1 Usage and interface (`provrml_parser`) 1047
221.2 Documentation on exports (`provrml_parser`) 1047
parser/2 (pred) 1047
nodeDeclaration/4 (pred) 1047
field_Id/1 (prop) 1048

222 parser_util (library) 1049

222.1 Usage and interface (`parser_util`) 1049
222.2 Documentation on exports (`parser_util`) 1049
at_least_one/4 (pred) 1049
at_least_one/5 (pred) 1049
fillout/4 (pred) 1050
fillout/5 (pred) 1050
create_node/3 (pred) 1051
create_field/3 (pred) 1051
create_field/4 (pred) 1051
create_field/5 (pred) 1051
create_directed_field/5 (pred) 1052
correct_commenting/4 (pred) 1052
create_parse_structure/1 (pred) 1053
create_parse_structure/2 (pred) 1053
create_parse_structure/3 (pred) 1053
create_environment/4 (pred) 1054
insert_comments_in_beginning/3 (pred) 1054

get_environment_name/2 (pred) 1054
get_environment_type/2 (pred) 1055
get_row_number/2 (pred) 1055
add_environment_whitespace/3 (pred) 1055
get_indentation/2 (pred) 1056
inc_indentation/2 (pred) 1056
dec_indentation/2 (pred) 1056
add_indentation/3 (pred) 1056
reduce_indentation/3 (pred) 1057
push_whitespace/3 (pred) 1057
push_dictionaries/3 (pred) 1057
get_parsed/2 (pred) 1057
get_environment/2 (pred) 1058
inside_proto/1 (pred) 1058
get_dictionaries/2 (pred) 1058
strip_from_list/2 (pred) 1058
strip_from_term/2 (pred) 1059
strip_clean/2 (pred) 1059
strip_exposed/2 (pred) 1059
strip_restricted/2 (pred) 1059
strip_interface/2 (pred) 1059
set_parsed/3 (pred) 1060
set_environment/3 (pred) 1060
insert_parsed/3 (pred) 1060
reverse_parsed/2 (pred) 1061
stop_parse/2 (pred) 1061
look_first_parsed/2 (pred) 1061
get_first_parsed/3 (pred) 1061
remove_code/3 (pred) 1062
look_ahead/3 (pred) 1062

223 possible (library) 1063

223.1 Usage and interface (`possible`) 1063
223.2 Documentation on exports (`possible`) 1063
continue/3 (pred) 1063

224 tokeniser (library) 1065

224.1 Usage and interface (`tokeniser`) 1065
224.2 Documentation on exports (`tokeniser`) 1065
tokeniser/2 (pred) 1065
token_read/3 (pred) 1065

225 Pattern (regular expression) matching 1069

225.1 Usage and interface (`regexp_doc`) 1069
225.2 Documentation on internals (`regexp_doc`) 1069
match_shell/3 (pred) 1069
match_shell/2 (pred) 1070
match_posix/2 (pred) 1070
match_posix/4 (pred) 1070
match_posix_rest/3 (pred) 1070
match_posix_matches/3 (pred) 1070
match_struct/4 (pred) 1071
match_pred/2 (pred) 1071
replace_first/4 (pred) 1071
replace_all/4 (pred) 1071

226 regexp_code (library) 1073
226.1 Usage and interface (`regexp_code`) 1073
226.2 Documentation on exports (`regexp_code`) 1073
match_shell/3 (pred) 1073
match_shell/2 (pred) 1073
match_posix/2 (pred) 1073
match_posix/4 (pred) 1074
match_posix_rest/3 (pred) 1074
match_posix_matches/3 (pred) 1074
match_struct/4 (pred) 1075
match_pred/2 (pred) 1075
replace_first/4 (pred) 1075
replace_all/4 (pred) 1075
shell_regexp/1 (regtype) 1075
posix_regexp/1 (regtype) 1075
struct_regexp/1 (regtype) 1076
226.3 Documentation on multifiles (`regexp_code`) 1076
define_flag/3 (pred) 1076

227 Automatic tester 1077
227.1 Usage and interface (`tester`) 1077
227.2 Documentation on exports (`tester`) 1077
run_tester/10 (pred) 1077
227.3 Other information (`tester`) 1078
227.3.1 Understanding run_test predicate 1078
227.3.2 More complex example 1079

228 Measuring features from predicates (time cost or memory used) 1083
228.1 Usage and interface (`time_analyzer`) 1083
228.2 Documentation on exports (`time_analyzer`) 1083
performance/3 (pred) 1083
benchmark/6 (pred) 1084
compare_benchmark/7 (pred) 1084
generate_benchmark_list/7 (pred) 1085
benchmark2/6 (pred) 1085
compare_benchmark2/7 (pred) 1085
generate_benchmark_list2/7 (pred) 1086
sub_times/3 (pred) 1086
div_times/2 (pred) 1086
cost/3 (pred) 1086
generate_plot/3 (udreexp) 1087
generate_plot/2 (udreexp) 1087
set_general_options/1 (udreexp) 1087
get_general_options/1 (udreexp) 1087

229 XDR handle library 1089

229.1 Usage and interface (`xdr_handle`) 1089
229.2 Documentation on exports (`xdr_handle`) 1089
xdr_tree/3 (pred) 1089
xdr_tree/1 (pred) 1089
xdr_node/1 (regtype) 1090
xdr2html/4 (pred) 1090
xdr2html/2 (pred) 1090
unfold_tree/2 (pred) 1090
unfold_tree_dic/3 (pred) 1091
xdr_xpath/2 (pred) 1091

230 XML query library 1093

230.1 Usage and interface (`xml_path_doc`) 1093
230.2 Documentation on exports (`xml_path_doc`) 1094
xml_search/3 (pred) 1094
xml_parse/3 (pred) 1094
xml_parse_match/3 (pred) 1094
xml_search_match/3 (pred) 1095
xml_index_query/3 (pred) 1095
xml_index_to_file/2 (pred) 1095
xml_index/1 (pred) 1095
xml_query/3 (pred) 1096
230.3 Documentation on internals (`xml_path_doc`) 1096
canonic_xml_term/1 (regtype) 1096
canonic_xml_item/1 (regtype) 1096
tag_attrib/1 (regtype) 1096
canonic_xml_query/1 (regtype) 1096
canonic_xml_subquery/1 (regtype) 1097

PART XII - Appendices 1099

231 Installing Ciao from the source distribution 1101

231.1 Un*x installation summary 1101
231.2 Un*x full installation instructions 1102
231.3 Checking for correct installation on Un*x 1105
231.4 Cleaning up the source directory 1106
231.5 Multiarchitecture support 1106
231.6 Installation and compilation under Windows 1107
231.7 Porting to currently unsupported operating systems 1107
231.8 Troubleshooting (nasty messages and nifty workarounds) .. 1108

232 Installing Ciao from a Win32 binary distribution 1111

232.1 Win32 binary installation summary 1111
232.2 Checking for correct installation on Win32 1112
232.3 Compiling the miscellaneous utilities under Windows 1113
232.4 Server installation under Windows 1113
232.5 CGI execution under IIS 1113
232.6 Uninstallation under Windows 1114

233 Beyond installation 1115
233.1 Architecture-specific notes and limitations 1115
233.2 Keeping up to date with the Ciao users mailing list 1115
233.3 Downloading new versions 1115
233.4 Reporting bugs 1116

References 1117

Library/Module Index 1125

Predicate/Method Index 1127

Property Index 1129

Regular Type Index 1131

Declaration Index 1133

Concept Index 1135

Author Index 1137

Global Index 1139

Summary

Ciao is a *public domain*, *next generation* multi-paradigm programming environment with a unique set of features:

- **Ciao** offers a complete Prolog system, supporting *ISO-Prolog*, but its novel modular design allows both *restricting* and *extending* the language. As a result, it allows working with *fully declarative subsets* of Prolog and also to *extend* these subsets (or ISO-Prolog) both syntactically and semantically. Most importantly, these restrictions and extensions can be activated separately on each program module so that several extensions can coexist in the same application for different modules.
- **Ciao** also supports (through such extensions) programming with functions, higher-order (with predicate abstractions), constraints, and objects, as well as feature terms (records), persistence, several control rules (breadth-first search, iterative deepening, ...), concurrency (threads/engines), a good base for distributed execution (agents), and parallel execution. Libraries also support WWW programming, sockets, external interfaces (C, Java, TclTk, relational databases, etc.), etc.
- **Ciao** offers support for *programming in the large* with a robust module/object system, module-based separate/incremental compilation (automatically –no need for makefiles), an assertion language for declaring (*optional*) program properties (including types and modes, but also determinacy, non-failure, cost, etc.), automatic static inference and static/dynamic checking of such assertions, etc.
- **Ciao** also offers support for *programming in the small* producing small executables (including only those libraries actually used by the program) and support for writing scripts.
- The **Ciao** programming environment includes a classical top-level and a rich emacs interface with an embeddable source-level debugger and a number of execution visualization tools.
- The **Ciao** compiler (which can be run outside the top level shell) generates several forms of architecture-independent and stand-alone executables, which run with speed, efficiency, and executable size which are very competitive with other commercial and academic languages (including other Prolog/CLP systems). Library modules can be compiled into compact bytecode or C source files, and linked statically, dynamically, or autoloaded.
- The novel modular design of **Ciao** enables, in addition to modular program development, effective global program analysis and static debugging and optimization via source to source program transformation. These tasks are performed by the **Ciao preprocessor** (`ciaopp`, distributed separately).
- The **Ciao** programming environment also includes `lpdoc`, an automatic documentation generator for LP/CLP programs. It processes source files adorned with (**Ciao**) assertions and machine-readable comments and generates manuals in many formats including `postscript`, `pdf`, `texinfo`, `info`, `HTML`, `man`, etc. , as well as on-line help, ascii `README` files, entries for indices of manuals (`info`, WWW, ...), and maintains WWW distribution sites.

Ciao is distributed under the GNU Library General Public License (LGPL).

This documentation corresponds to version 1.13#1 (2011/3/15, 20:8:35 CEST).

1 Introduction

1.1 About this manual

This is the *Reference Manual* for the Ciao development system. It contains basic information on how to install Ciao and how to write, debug, and run Ciao programs from the command line, from inside GNU `emacs`, or from a windowing desktop. It also documents all the libraries available in the standard distribution.

This manual has been generated using the *LPdoc* semi-automatic documentation generator for LP/CLP programs [HC97,Her00]. `lpdoc` processes Ciao files (and files in Prolog and other CLP languages) adorned with assertions and machine-readable comments, which should be written in the Ciao assertion language [PBH97,PBH00]. From these, it generates manuals in many formats including `postscript`, `pdf`, `texinfo`, `info`, `HTML`, `man`, etc., as well as on-line help, ascii `README` files, entries for indices of manuals (`info`, WWW, ...), and maintains WWW distribution sites.

The big advantage of this approach is that it is easier to keep the on-line and printed documentation in sync with the source code [Knu84]. As a result, *this manual changes continually as the source code is modified*. Because of this, the manual has a version number. You should make sure the manual you are reading, whether it be printed or on-line, coincides with the version of the software that you are using.

The approach also implies that there is often a variability in the degree to which different libraries or system components are documented. Many libraries offer abundant documentation, but a few will offer little. The latter is due to the fact that we tend to include libraries in the manual if the code is found to be useful, even if they may still contain sparse documentation. This is because including a library in the manual will at the bare minimum provide formal information (such as the names of exported predicates and their arity, which other modules it loads, etc.), create index entries, pointers for on-line help in the electronic versions of the manuals, and command-line completion capabilities inside `emacs`. Again, the manual is being updated continuously as the different libraries (and machine-readable documentation in them) are improved.

1.2 About the Ciao development system

The Ciao system is a full programming environment for developing programs in the Prolog language and in several other languages which are extensions and modifications of Prolog and (Constraint) Logic Programming in several interesting and useful directions. The programming environment offers a number of tools such as the Ciao standalone compiler (`ciaoc`), a traditional-style top-level interactive shell (`ciaosh` or `ciao`), an interpreter of scripts written in Ciao (`ciao-shell`), a Ciao (and Prolog) `emacs` mode (which greatly helps the task of developing programs with support for editing, debugging, version/change tracking, etc.), numerous libraries, a powerful program preprocessor (`ciaopp` [BGH99,BLGPH04,HBPLG99], which supports static debugging and optimization from program analysis via source to source program transformation), and an automatic documentation generator (`lpdoc`) [HC97,Her00]. A number of execution visualization tools [CGH93,CH00d,CH00c] are also available.

This manual documents the first four of the tools mentioned above [see PART I - The program development environment], and the Ciao language and libraries. The `ciaopp` and `lpdoc` tools are documented in separate manuals.

The Ciao language [see PART II - The Ciao basic language (engine)] has been designed from the ground up to be small, but to also allow extensions and restrictions in a modular way. The first objective allows producing small executables (including only those builtins used by the program), providing basic support for pure logic programming, and being able to write scripts

in Ciao. The second one allows supporting standard ISO-Prolog [see PART III - ISO-Prolog library (iso)], as well as powerful extensions such as constraint logic programming, functional logic programming, and object-oriented logic programming [see PART VII - Ciao extensions], and restrictions such as working with pure horn clauses.

The design of Ciao has also focused on allowing modular program development, as well as automatic program manipulation and optimization. Ciao includes a robust module system [CH00a], module-based automatic incremental compilation [CH99b], and modular global program analysis, debugging and optimization [PH99], based on a rich assertion language [see PART V - Assertions, Properties, Types, Modes, Comments (assertions)] for declaring (optional) program properties (including types and modes), which can be checked either statically or dynamically. The program analysis, static debugging and optimization tasks related to these assertions are performed by the `ciaopp` preprocessor, as mentioned above. These assertions (together with special comment-style declarations) are also the ones used by the `lpdoc` autodocumenter to generate documentation for programs (the comment-style declarations are documented in the `lpdoc` manual).

Ciao also includes several other features and utilities, such as support for several forms of executables, concurrency (threads), distributed and parallel execution, higher-order, WWW programming (PiLLoW [CHV96b]), interfaces to other languages like C and Java, database interfaces, graphical interfaces, etc., etc. [see PARTS VI to XI].

1.3 ISO-Prolog compliance versus extensibility

One of the innovative features of Ciao is that it has been designed to subsume *ISO-Prolog* (International Standard ISO/IEC 13211-1, PROLOG: Part 1–General Core [DEDC96]), while at the same time extending it in many important ways. The intention is to ensure that all ISO-compliant Prolog programs run correctly under Ciao. At the same time, the Ciao module system (see [PART II - The Ciao basic language (engine)] and [CH00a] for a discussion of the motivations behind the design) allows selectively avoiding the loading of most ISO-builtins (and changing some other ISO characteristics) when not needed, so that it is possible to work with purer subsets of Prolog and also to build small executables. Also, this module system makes it possible to develop extensions using these purer subsets (or even the full ISO-standard) as a starting point. Using these features, the Ciao distribution includes libraries which significantly extend the language both syntactically and semantically.

Compliance with ISO is still not complete: currently there are some minor deviations in, e.g., the treatment of characters, the syntax, some of the arithmetic functions, and part of the error system. On the other hand, Ciao has been reported by independent sources (members of the standarization body) to be one of the most conforming Prologs at the moment of this writing, and the first one to be able to compile all the standard-conforming test cases. Also, Ciao does not offer a strictly conforming mode which rejects uses of non-ISO features. However, in order to aid programmers who wish to write standard compliant programs, library predicates that correspond to those in the ISO-Prolog standard are marked specially in the manuals, and differences between the Ciao and the prescribed ISO-Prolog behaviours, if any, are commented appropriately.

The intention of the Ciao developers is to progressively complete the compliance of Ciao with the published parts of the ISO standard as well as with other reasonable extensions of the standard may be published in the future. However, since one of the design objectives of Ciao is to address some shortcomings of previous implementations of Prolog and logic programming in general, we also hope that some of the better ideas present in the system will make it eventually into other systems and the standards.

1.4 About the name of the System

Ciao is often referred to as "Ciao Prolog," a name which has an interesting (and not unintended) interpretation. Ciao is an interesting word which means both *hello* and *goodbye*. 'Ciao Prolog' intends to be a really good, all-round, freely available ISO-Prolog system which can be used as a classical Prolog, in both academic and industrial environments (and, in particular, to introduce users to Prolog and to constraint and logic programming –the *hello* part). An indeed many programmers use it just that way. But Ciao is also a new-generation, multiparadigm programming language and program development system which goes well beyond Prolog and other classical logic programming languages. And it has the advantage (when compared to other new-generation LP systems) that it does so while keeping full Prolog compatibility when needed.

1.5 Referring to Ciao

If you find Ciao or any of its components useful, we would appreciate very much if you added a reference to this manual (i.e., the Ciao reference manual [BCC97]) in your work. The following is an appropriate BiBTeX entry with the relevant data:

```
@techreport{ciao-reference-manual-tr,
  author =       {F. Bueno and D. Cabeza and M. Carro and M. Hermenegildo
                  and P. L\'{o}pez-Garc\'{\i}a and G. Puebla},
  title =        {The Ciao System. Reference Manual},
  institution =  {School of Computer Science, T. U. of Madrid (UPM)
                  and IMDEA Software Institute},
  year =         1997,
  month =        {August},
  number =       {{CLIP}3/1997.2011},
  note =         {Available from http://www.cliplab.org/}
}
```

1.6 Syntax terminology and notational conventions

This manual assumes some familiarity with logic programming and the Prolog language. The reader is referred to standard textbooks on logic programming and Prolog (such as, e.g., [SS86,CM81,Apt97,Hog84]) for background. However, we would like to refresh herein some concepts for the sake of establishing terminology. Also, we will briefly introduce a few of the extensions that Ciao brings to the Prolog language.

1.6.1 Predicates and their components

Procedures are called *predicates* and predicate calls *literals*. They all have the classical syntax of procedures (and of logic predications and of mathematical functions). Predicates are identified in this manual by a keyword 'PREDICATE' at the right margin of the place where they are documented.

Program instructions are expressions made up of control constructs (Chapter 13 [Control constructs/predicates], page 99) and literals, and are called *goals*. Literals are also (atomic) goals.

A predicate definition is a sequence of clauses. A clause has the form "`H :- B.`" (ending in '`.`'), where `H` is syntactically the same as a literal and is called the clause *head*, and `B` is a goal and is called the clause *body*. A clause with no body is written "`H.`" and is called a *fact*. Clauses with body are also called *rules*. A program is a sequence of predicate definitions.

1.6.2 Characters and character strings

We adopt the following convention for delineating character strings in the text of this manual: when a string is being used as an atom it is written thus: `user` or `'user'`; but in all other circumstances double quotes are used (as in `"hello"`).

When referring to keyboard characters, printing characters are written thus: ⟨a⟩, while control characters are written like this: ⟨^A⟩. Thus ⟨^C⟩ is the character you get by holding down the ⟨CTL⟩ key while you type ⟨c⟩. Finally, the special control characters carriage-return, line-feed and space are often abbreviated to ⟨RET⟩, ⟨LFD⟩ and ⟨SPC⟩ respectively.

1.6.3 Predicate specs

Predicates are distinguished by their name *and* their arity. We will call `name/arity` a *predicate spec*. The notation `name/arity` is therefore used when it is necessary to refer to a predicate unambiguously. For example, `concatenate/3` specifies the predicate which is named "concatenate" and which takes 3 arguments.

(Note that different predicates may have the same name and different arity. Conversely, of course, they may have the same arity and different name.)

1.6.4 Modes

When documenting a predicate, we will often describe its usage with a mode spec which has the form `name(Arg1, ..., ArgN)` where each `Arg` may be preceded by a *mode*. A mode is a functor which is wrapped around an argument (or prepended if defined as an operator). Such a mode allows documenting in a compact way the instantiation state on call and exit of the argument to which it is applied. The set of modes which can be used in Ciao is not fixed. Instead, arbitrary modes can be defined by in programs using the `modedef/1` declarations of the Ciao *assertion language* (Chapter 54 [The Ciao assertion package], page 339 for details). Modes are identified in this manual by a keyword 'MODE'.

Herein, we will use the set of modes defined in the Ciao `isomodes` library, which is essentially the same as those used in the ISO-Prolog standard (Chapter 58 [ISO-Prolog modes], page 371).

1.6.5 Properties and types

Although Ciao is *not* a typed language, it allows writing (and using) types, as well as (more general) properties. There may be properties of the states and of the computation. Properties of the states allow expressing characteristics of the program variables during computation, like in `sorted(X)` (`X` is a sorted list). Properties of the computation allow expressing characteristics of a whole computation, like in `is_det(p(X,Y))` (such calls yield only one solution). Properties are just a special form of predicates (Chapter 56 [Declaring regular types], page 355) and are identified in this manual by a keyword 'PROPERTY'.

Ciao types are *regular types* (Chapter 56 [Declaring regular types], page 355), which are a special form of properties themselves. They are identified in this manual by a keyword 'REGTYPE'.

1.6.6 Declarations

A *declaration* provides information to one of the Ciao environment tools. Declarations are interspersed in the code of a program. Usually the target tool is either the compiler (telling it that a predicate is dynamic, or a meta-predicate, etc.), the preprocessor (which understands declarations of properties and types, assertions, etc.), or the autodocumenter (which understands the previous declarations and also certain "comment" declarations).

A declaration has the form `:- D.` where `D` is syntactically the same as a literal. Declarations are identified in this manual by a keyword 'DECLARATION'.

In Ciao users can define (and document) new declarations. New declarations are typically useful when defining extensions to the language (which in Ciao are called packages). Such extensions are often implemented as expansions (see Chapter 26 [Extending the syntax], page 193). There are many such extensions in Ciao. The `functions` library, which provides fuctional syntax, is an example. The fact that in Ciao expansions are local to modules (as operators, see below) makes it possible to use a certain language extension in one module without affecting other modules.

1.6.7 Operators

An *operator* is a functor (or predicate name) which has been declared as such, thus allowing its use in a prefix, infix, or suffix fashion, instead of the standard procedure-like fashion. E.g., declaring `+` as an infix operator allows writing `X+Y` instead of `'+'(X,Y)` (which may still, of course, be written).

Operators in Ciao are local to the module/file where they are declared. However, some operators are standard and allowed in every program (see Chapter 37 [Defining operators], page 255). This manual documents the operator declarations in each (library) module where they are included. As with expansions, the fact that in Ciao operators are local to modules makes it possible to use a certain language extension in one module without affecting other modules.

1.7 A tour of the manual

The rest of the introductory chapters after this one provide a first "getting started" introduction for newcomers to the Ciao system. The rest of the chapters in the manual are organized into a sequence of major parts as follows:

1.7.1 PART I - The program development environment

This part documents the components of the basic Ciao program development environment. They include:

ciaoc: the standalone compiler, which creates executables without having to enter the interactive top-level.

ciaosh: (also invoked simply as `ciao`) is an interactive top-level shell, similar to the one found on most Prolog systems (with some enhancements).

debugger.pl: a Byrd box-type debugger, similar to the one found on most Prolog systems (also with some enhancements, such as source-level debugging). This is not a standalone application, but is rather included in `ciaosh`, as is done in other systems supporting the Prolog language. However, it is also *embeddable*, in the sense that it can be included as a library in executables, and activated dynamically and conditionally while such executables are running.

ciao-shell: an interpreter/compiler for *Ciao scripts* (i.e., files containing Ciao code which run without needing explicit compilation).

ciao.el: a *complete program development enviroment*, based on GNU emacs, with syntax coloring, direct access to all the tools described above (as well as the preprocessor and the documenter), atomatic location of errors, source-level debugging, context-sensitive access to on-line help/manuals, etc. The use of this environment is *very highly recommended*!

The Ciao program development environment also includes `ciaopp`, the preprocessor, and `lpdoc`, the documentation generator, which are described in separate manuals.

1.7.2 PART II - The Ciao basic language (engine)

This part documents the *Ciao basic builtins*. These predefined predicates and declarations are available in every program, unless the `pure` package is used (by using a `:- module(_,_,[pure]).` declaration or `:- use_package(pure).`). These predicates are contained in the `engine` directory within the `lib` library. The rest of the library predicates, including the packages that provide most of the ISO-Prolog builtins, are documented in subsequent parts.

1.7.3 PART III - ISO-Prolog library (iso)

This part documents the *iso* package which provides to Ciao programs (most of) the ISO-Prolog functionality, including the *ISO-Prolog builtins* not covered by the basic library.

1.7.4 PART IV - Classic Prolog library (classic)

This part documents some Ciao libraries which provide additional predicates and functionalities that, despite not being in the ISO standard, are present in many popular Prolog systems. This includes definite clause grammars (DCGs), "Quintus-style" internal database, list processing predicates, DEC-10 Prolog-style input/output, formatted output, dynamic loading of modules, activation of operators at run-time, etc.

1.7.5 PART V - Assertions, Properties, Types, Modes, Comments (assertions)

Ciao allows *annotating* the program code with *assertions*. Such assertions include type and instantiation mode declarations, but also more general properties as well as comments for *autodocumentation* in the *literate programming* style. These assertions document predicates (and modules and whole applications) and can be used by the Ciao preprocessor/compiler while debugging and optimizing the program or library, and by the Ciao documenter to build program or library reference manuals.

1.7.6 PART VI - Ciao library miscellanea

This part documents several Ciao libraries which provide different useful additional functionality. Such functionality includes performing operating system calls, gathering statistics from the Ciao engine, file and filename manipulation, error and exception handling, fast reading and writing of terms (marshalling and unmarshalling), file locking, issuing program and error messages, pretty-printing programs and assertions, a browser of the system libraries, additional expansion utilities, concurrent aggregates, graph visualization, etc.

1.7.7 PART VII - Ciao extensions

The libraries documented in this part extend the Ciao language in several different ways. The extensions include:

- pure Prolog programming (well, this can be viewed more as a restriction than an extension);
- feature terms or *records* (i.e., structures with names for each field);
- parallel programming (e.g., &-Prolog style);
- functional syntax;
- higher-order;
- global variables;
- `setarg` and `undo`;
- delaying predicate execution;

- active modules;
- breadth-first execution;
- iterative deepening-based execution;
- constraint logic programming;
- object oriented programming.

1.7.8 PART VIII - Interfaces to other languages and systems

The following interfaces to/from Ciao are documented in this part:

- External interface (e.g., to C).
- Socket interface.
- Tcl/tk interface.
- Web interface (http, html, xml, etc.);
- Persistent predicate databases (interface between the Ciao internal database and the external file system).
- SQL-like database interface (interface between the Ciao internal database and external SQL/ODBC systems).
- Java interface.
- Calling emacs from Ciao.

1.7.9 PART IX - Abstract data types

This part includes libraries which implement some generic data structures (abstract data types) that are used frequently in programs or in the Ciao system itself.

1.7.10 PART X - Miscellaneous standalone utilities

This is the documentation for a set of miscellaneous standalone utilities contained in the `etc` directory of the Ciao distribution.

1.7.11 PART XI - Contributed libraries

This part includes a number of libraries which have contributed by users of the Ciao system. Over time, some of these libraries are moved to the main library directories of the system.

1.7.12 PART XII - Appendices

These appendices describe the installation of the Ciao environment on different systems and some other issues such as reporting bugs, signing up on the Ciao user's mailing list, downloading new versions, limitations, etc.

1.8 Acknowledgments

The Ciao system is a joint effort on one side of some present (Francisco Bueno, Manuel Carro, Manuel Hermenegildo, *Pedro López*, and *Germán Puebla*) and past (Daniel Cabeza, *María José García de la Banda*) members of the *CLIP group* at the School of Computer Science, *Technical University of Madrid* and at the *IMDEA Software Institute*, and on the other side of several colleagues and students that have collaborated with us over the years of its development. The following is an (inevitably incomplete) list of those that have contributed most significantly to the development of Ciao:

- The *Ciao engine*, compiler, preprocessor, libraries, and documentation, although completely rewritten at this point, have their origins in the *&-Prolog parallel Prolog* engine and *parallelizing compiler*, developed by Manuel Hermenegildo, Kevin Greene, Kalyan Muthukumar, and Roger Nasr at *MCC* and later at *UPM*. The &-Prolog engine and low-level (*WAM*) compilers in turn were derived from early versions (0.5 to 0.7) of *SICStus Prolog* [Car88]. SICStus is an excellent, high performance Prolog system, developed by Mats Carlsson and colleagues at the *Swedish Institute of Computer Science* (*SICS*), that every user of Prolog should check out [Swe95,AAF91]. Very special thanks are due to Seif Haridi, Mats Carlsson, and colleagues at SICS for allowing the SICStus 0.5-0.7 components in *&-Prolog* and its successor, Ciao, to be distributed freely. Parts of the parallel abstract machine have been developed in collaboration with Gopal Gupta and Enrico Pontelli (*New Mexico State University*).
- Many aspects of the analyzers in the *Ciao preprocessor* (`ciaopp`) have been developed in collaboration with Peter Stuckey (*Melbourne U.*), Kim Marriott (*Monash U.*), Maurice Bruynooghe, Gerda Janssens, Anne Mulkers, and Veroniek Dumortier (*K.U. Leuven*), and Saumya Debray (*U. of Arizona*). The assertion system has been developed in collaboration with Jan Maluzynski and Wlodek Drabent (*Linkoping U.*) and Pierre Deransart (*INRIA*). The core of type inference system derives from the system developed by John Gallagher [GdW94] (*Bristol University*) and later adapted to CLP(FD) by Pawel Pietrzak (*Linkoping U.*).
- The constraint solvers for R and Q are derived from the code developed by Christian Holzbauer (*Austrian Research Institute for AI* in Vienna) [Hol94,Hol90,Hol92].
- The Ciao manuals include material from the *DECsystem-10 Prolog User's Manual* by D.L. Bowen (editor), L. Byrd, F.C.N. Pereira, L.M. Pereira, and D.H.D. Warren [BBP81]. They also contain material from the SICStus Prolog user manuals for SICStus versions 0.5-0.7 by Mats Carlsson and Johan Widen [Car88], as well as from the Prolog ISO standard documentation [DEDC96].
- Ciao is designed to be highly extendable in a modular way. Many of the libraries distributed with Ciao have been developed by other people all of which is impossible to mention here. Individual author names are included in the documentation of each library and appear in the indices.
- The development of the Ciao system has been supported in part by European research projects PEPMA, ACCLAIM, PARFORCE, DISCIPL, AMOS, ASAP, MOBIUS, and SCUBE; by MICYT projects IPL-D, ELLA, EDIPIA, CUBICO, MERIT, and DOVES; and by CM projects PROMESAS and PROMETIDOS.

If you feel you have contributed to the development of Ciao and we have forgotten to add your name to this list or to the acknowledgements given in the different chapters and indices, please let us know and we will be glad to give proper credits.

1.9 Version/Change Log

Version 1.12 (2005/7/3, 18:50:50 CEST)

Temporary version before transition to SVN log comments. (Jose Morales)

Version 1.10 (2004/7/29, 16:12:3 CEST)

- Classical prolog mode as default behavior.
- Emacs-based environment improved.
  - Improved emacs inferior (interaction) mode for Ciao and CiaoPP.
  - Xemacs compatibility improved (thanks to A. Rigo).
  - New icons and modifications in the environment for the preprocessor.
  - Icons now installed in a separate dir.

    - Compatibility with newer versions of `Cygwin`.
    - Changes to programming environment:
        - Double-click startup of programming environment.
        - Reorganized menus: help and customization grouped in separate menus.
        - Error location extended.
        - Automatic/Manual location of errors produced when running Ciao tools now customizable.
        - Presentation of CiaoPP preprocessor output improved.
    - Faces and coloring improved:
        - Faces for syntax-based highlighting more customizable.
        - Syntax-based coloring greatly improved. Literal-level assertions also correctly colored now.
        - Syntax-based coloring now also working on ASCII terminals (for newer versions of emacs).
        - Listing user-defined directives allowed to be colored in special face.
        - Syntax errors now colored also in inferior buffers.
        - Customizable faces now appear in the documentation.
        - Added new tool bar button (and binding) to refontify block/buffer.
        - Error marks now cleared automatically also when generating docs.
        - Added some fixes to hooks in lpdoc buffer.
- Bug fixes in compiler.
    - Replication of clauses in some cases (thanks to S. Craig).
- Improvements related to supported platforms
    - Compilation and installation in different palatforms have been improved.
    - New Mac OS X kernels supported.
- Improvement and bugs fixes in the engine:
    - Got rid of several segmentation violation problems.
    - Number of significant decimal digits to be printed now computed accurately.
    - Added support to test conversion of a Ciao integer into a machine int.
    - Unbound length atoms now always working.
    - C interface .h files reachable through a more standard location (thanks to R. Bagnara).
    - Compatibility with newer versions of gcc.
- New libraries and utilities added to the system:
    - Factsdb: facts defined in external files can now be automatically cached on-demand.
    - Symfnames: File aliasing to internal streams added.
- New libraries added (in beta state):
    - fd: clp(FD)
    - xml_path: XML querying and transformation to Prolog.
    - xdr_handle: XDR schema to HTML forms utility.
    - ddlist: Two-way traversal list library.
    - gnuplot: Interface to GnuPlot.

    - time_analyzer: Execution time profiling.
- Some libraries greatly improved:
    - Interface to Tcl/Tk very improved.
        - Corrected many bugs in both interaction Prolog to Tcl/Tk and viceversa.
        - Execution of Prolog goals from TclTk revamped.
        - Treatment of Tcl events corrected.
        - Predicate `tcl_eval/3` now allows the execution of Tcl procedures running multiple Prolog goals.
        - Documentation heavily reworked.
        - Fixed unification of prolog goals run from the Tcl side.
    - Pillow library improved in many senses.
        - HTTP media type parameter values returned are always strings now, not atoms.
        - Changed verbatim() pillow term so that newlines are translated to <br>.
        - Changed management of cookies so that special characters in values are correctly handled.
        - Added predicate `url_query_values/2`, reversible. Predicate `url_query/2` now obsolete.
        - Now attribute values in tags are escaped to handle values which have double quotes.
        - Improved `get_form_input/1` and `url_query/2` so that names of parameters having unusual characters are always correctly handled.
    - Fixed bug in tokenizer regarding non-terminated single or multiple-line comments. When the last line of a file has a single-line comment and does not end in a newline, it is accepted as correct. When an open-comment /* sequence is not terminated in a file, a syntax error exception is thrown.
- Other libraries improved:
    - Added native_props to assertions package and included `nonground/1`.
    - In atom2terms, changed interpretation of double quoted strings so that they are not parsed to terms.
    - Control on exceptions improved.
    - Added `native/1,2` to basic_props.
    - Davinci error processing improved.
    - Foreign predicates are now automatically declared as implementation-defined.
    - In lists, added `cross_product/2` to compute the cartesian product of a list of lists. Also added `delete_non_ground/3`, enabling deletion of nonground terms from a list.
    - In llists added `transpose/2` and changed `append/2` implementation with a much more efficient code.
    - The make library has been improved.
    - In persdb, added `pretractall_fact/1` and `retractall_fact/1` as persdb native capabilities.
    - Improved behavior with user environment from persdb.

- In persdb, added support for `persistent_dir/4`, which includes arguments to specify permission modes for persistent directory and files.
- Some minor updates in persdb_sql.
- Added treatment of operators and module:pred calls to pretty-printer.
- Updated report of read of syntax errors.
- File locking capabilities included in `open/3`.
- Several improvements in library system.
- New input/output facilities added to sockets.
- Added `most_specific_generalization/3` and `most_general_instance/3` to terms_check.
- Added `sort_dict/2` to library vndict.
- The xref library now treats also empty references.

- Miscellaneous updates:
  - Extended documentation in libraries actmods, arrays, foreign_interface, javall, persdb_mysql, prolog_sys, old_database, and terms_vars.

Version 1.9 (2002/5/16, 23:17:34 CEST)
New development version after stable 1.8p0 (MCL, DCG)

Version 1.8 (2002/5/16, 21:20:27 CEST)

- Improvements related to supported platforms:
  - Support for Mac OS X 10.1, based on the Darwin kernel.
  - Initial support for compilation on Linux for Power PC (contributed by Paulo Moura).
  - Workaround for incorrect C compilation while using newer (> 2.95) gcc compilers.
  - .bat files generated in Windows.
- Changes in compiler behavior and user interface:
  - Corrected a bug which caused wrong code generation in some cases.
  - Changed execution of initialization directives. Now the initialization of a module/file never runs before the initializations of the modules from which the module/file imports (excluding circular dependences).
  - The engine is more intelligent when looking for an engine to execute bytecode; this caters for a variety of situations when setting explicitly the CIAOLIB environment variable.
  - Fixed bugs in the toplevel: behaviour of `module:main` calls and initialization of a module (now happens after related modules are loaded).
  - Layout char not needed any more to end Prolog files.
  - Syntax errors now disable .itf creation, so that they show next time the code is used without change.
  - Redefinition warnings now issued only when an unqualified call is seen.
  - Context menu in Windows can now load a file into the toplevel.
  - Updated Windows installation in order to run CGI executables under Windows: a new information item is added to the registry.
  - Added new directories found in recent Linux distributions to INFOPATH.
  - Emacs-based environment and debugger improved:
    - Errors located immediataly after code loading.
    - Improved ciao-check-types-modes (preprocessor progress now visible).

  - Fixed loading regions repeatedly (no more predicate redefinition warnings).
  - Added entries in `ciaopp` menu to set verbosity of output.
  - Fixed some additional xemacs compatibility issues (related to searches).
  - Errors reported by inferior processes are now explored in forward order (i.e., the first error rewported is the first one highlighted). Improved tracking of errors.
  - Specific tool bar now available, with icons for main fuctions (works from emacs 21.1 on). Also, other minor adaptations for working with emacs 21.1 and later.
  - Debugger faces are now locally defined (and better customization). This also improves comtability with xemacs (which has different faces).
  - Direct access to a common use of the preprocessor (checking modes/types and locating errors) from toolbar.
  - Inferior modes for Ciao and CiaoPP improved: contextual help turned on by default.
  - Fixes to set-query. Also, previous query now appears in prompt.
  - Improved behaviour of stored query.
  - Improved behaviour of recentering, finding errors, etc.
  - Wait for prompt has better termination characteristics.
  - Added new interactive entry points (M-x): ciao, prolog, ciaopp.
  - Better tracking of last inferior buffer used.
  - Miscellanous bugs removed; some colors changed to adapt to different Emacs versions.
  - Fixed some remaining incompatibilities with xemacs.
  - `:- doc` now also supported and highlighted.
  - Eliminated need for calendar.el
  - Added some missing library directives to fontlock list, organized this better.
- New libraries added to the system:
  - hiord: new library which needs to be loaded in order to use higher-order call/N and P(X) syntax. Improved model for predicate abstractions.
  - fuzzy: allows representing fuzzy information in the form or Prolog rules.
  - use_url: allows loading a module remotely by using a WWW address of the module source code
  - andorra: alternative search method where goals which become deterministic at run time are executed before others.
  - iterative deepening (id): alternative search method which makes a depth-first search until a predetermined depth is reached. Complete but in general cheaper than breadth first.
  - det_hook: allows making actions when a deterministic situation is reached.
  - ProVRML: read VRML code and translate it into Prolog terms, and the other way around.
  - io_alias_redirection: change where stdin/stdout/stderr point to from within Ciao programs.
  - tcl_tk: an interface to Tcl/Tk programs.

    - tcl_tk_obj: object-based interface to Tcl/Tk graphical objects.
    - CiaoPP: options to interface with the CiaoPP Prolog preprocessor.
- Some libraries greatly improved:
    - WebDB: utilities to create WWW-based database interfaces.
    - Improved java interface implementation (this forced renaming some interface primitives).
    - User-transparent persistent predicate database revamped:
        - Implemented passerta_fact/1 (asserta_fact/1).
        - Now it is never necessary to explicitly call init_persdb, a call to initialize_db is only needed after dynamically defining facts of persistent_dir/2. Thus, pcurrent_fact/1 predicate eliminated.
        - Facts of persistent predicates included in the program code are now included in the persistent database when it is created. They are ignored in successive executions.
        - Files where persistent predicates reside are now created inside a directory named as the module where the persistent predicates are defined, and are named as F_A* for predicate F/A.
        - Now there are two packages: persdb and 'persdb/ll' (for low level). In the first, the standard builtins asserta_fact/1, assertz_fact/1, and retract_fact/1 are replaced by new versions which handle persistent data predicates, behaving as usual for normal data predicates. In the second package, predicates with names starting with 'p' are defined, so that there is not overhead in calling the standard builtins.
        - Needed declarations for persistent_dir/2 are now included in the packages.
    - SQL now works with mysql.
    - system: expanded to contain more predicates which act as interface to the underlying system / operating system.
- Other libraries improved:
    - xref: creates cross-references among Prolog files.
    - concurrency: new predicates to create new concurrent predicates on-the-fly.
    - sockets: bugs corrected.
    - objects: concurrent facts now properly recognized.
    - fast read/write: bugs corrected.
    - Added 'webbased' protocol for active modules: publication of active module address can now be made through WWW.
    - Predicates in library(dynmods) moved to library(compiler).
    - Expansion and meta predicates improved.
    - Pretty printing.
    - Assertion processing.
    - Module-qualified function calls expansion improved.
    - Module expansion calls goal expansion even at runtime.
- Updates to builtins (there are a few more; these are the most relevant):
    - Added a prolog_flag to retrieve the version and patch.
    - current_predicate/1 in library(dynamic) now enumerates non-engine modules, prolog_sys:current_predicate/2 no longer exists.
    - exec/* bug fixed.

    - srandom/1 bug fixed.
  - Updates for C interface:
    - Fixed bugs in already existing code.
    - Added support for creation and traversing of Prolog data structures from C predicates.
    - Added support for raising Prolog exceptions from C predicates.
    - Preliminary support for calling Prolog from C.
  - Miscellaneous updates:
    - Installation made more robust.
    - Some pending documentation added.
    - 'ciao' script now adds (locally) to path the place where it has been installed, so that other programs can be located without being explicitly in the $PATH.
    - Loading programs is somewhat faster now.
    - Some improvement in printing path names in Windows.

Version 1.7 (2000/7/12, 19:1:20 CEST)
Development version following even 1.6 distribution.

Version 1.6 (2000/7/12, 18:55:50 CEST)
- Source-level debugger in emacs, breakpts.
- Emacs environment improved, added menus for Ciaopp and LPDoc.
- Debugger embeddable in executables.
- Stand-alone executables available for UNIX-like operating systems.
- Many improvements to emacs interface.
- Menu-based interface to autodocumenter.
- Threads now available in Win32.
- Many improvements to threads.
- Modular clp(R) / clp(Q).
- Libraries implementing And-fair breadth-first and iterative deepening included.
- Improved syntax for predicate abstractions.
- Library of higher-order list predicates.
- Better code expansion facilities (macros).
- New delay predicates (when/2).
- Compressed object code/executables on demand.
- The size of atoms is now unbound.
- Fast creation of new unique atoms.
- Number of clauses/predicates essentially unbound.
- Delayed goals with freeze restored.
- Faster compilation and startup.
- Much faster fast write/read.
- Improved documentation.
- Other new libraries.
- Improved installation/deinstallation on all platforms.
- Many improvements to autodocumenter.
- Many bug fixes in libraries and engine.

Version 1.5 (1999/11/29, 16:16:23 MEST)
Development version following even 1.4 distribution.

Version 1.4 (1999/11/27, 19:0:0 MEST)
- Documentation greatly improved.
- Automatic (re)compilation of foreign files.
- Concurrency primitives revamped; restored &Prolog-like multiengine capability.
- Windows installation and overall operation greatly improved.
- New version of O'Ciao class/object library, with improved performance.
- Added support for "predicate abstractions" in call/N.
- Implemented reexportation through reexport declarations.
- Changed precedence of importations, last one is now higher.
- Modules can now implicitly export all predicates.
- Many minor bugs fixed.

Version 1.3 (1999/6/16, 17:5:58 MEST)
Development version following even 1.2 distribution.

Version 1.2 (1999/6/14, 16:54:55 MEST)
Temporary version distributed locally for extensive testing of reexportation and other 1.3 features.

Version 1.1 (1999/6/4, 13:30:37 MEST)
Development version following even 1.0 distribution.

Version 1.0 (1999/6/4, 13:27:42 MEST)
- Added Tcl/Tk interface library to distribution.
- Added push_prolog_flag/2 and pop_prolog_flag/1 declarations/builtins.
- Filename processing in Windows improved.
- Added redefining/1 declaration to avoid redefining warnings.
- Changed syntax/1 declaration to use_package/1.
- Added add_clause_trans/1 declaration.
- Changed format of .itf files such that a '+' stands for all the standard imports from engine, which are included in c_itf source internally (from engine(builtin_exports)). Further changes in itf data handling, so that once an .itf file is read in a session, the file is cached and next time it is needed no access to the file system is required.
- Many bugs fixed.

Version 0.9 (1999/3/10, 17:3:49 CET)
- Test version before 1.0 release. Many bugs fixed.

Version 0.8 (1998/10/27, 13:12:36 MET)
- Changed compiler so that only one pass is done, eliminated `.dep` files.
- New concurrency primitives.
- Changed assertion comment operator to #.
- Implemented higher-order with call/N.
- Integrated SQL-interface to external databases with persistent predicate concept.
- First implementation of object oriented programming package.
- Some bugs fixed.

Version 0.7 (1998/9/15, 12:12:33 MEST)

- Improved debugger capabilities and made easier to use.
- Simplified assertion format.
- New arithmetic functions added, which complete all ISO functions.
- Some bugs fixed.

Version 0.6 (1998/7/16, 21:12:7 MET DST)

- Defining other path aliases (in addition to 'library') which can be loaded dynamically in executables is now possible.
- Added the posibility to define multifile predicates in the shell.
- Added the posibility to define dynamic predicates dynamically.
- Added addmodule meta-argument type.
- Implemented persistent data predicates.
- New version of PiLLoW WWW library (XML, templates, etc.).
- Ported active modules from "distributed Ciao" (independent development version of Ciao).
- Implemented lazy loading in executables.
- Modularized engine(builtin).
- Some bugs fixed.

Version 0.5 (1998/3/23)

- First Windows version.
- Integrated debugger in toplevel.
- Implemented DCG's as (Ciao-style) expansions.
- Builtins renamed to match ISO-Prolog.
- Made ISO the default syntax/package.

Version 0.4 (1998/2/24)

- First version with the new Ciao emacs mode.
- Full integration of concurrent engine and compiler/library.
- Added new_declaration/1 directive.
- Added modular syntax enhancements.
- Shell script interpreter separated from toplevel shell.
- Added new compilation warnings.

Version 0.3 (1997/8/20)

- Ciao builtins modularized.
- New prolog flags can be defined by libraries.
- Standalone comand-line compiler available, with automatic `"make"`.
- Added assertions and regular types.
- First version using the automatic documentation generator.

Version 0.2 (1997/4/16)

- First module system implemented.
- Implemented exceptions using catch/3 and throw/1.
- Added functional & record syntax.
- Added modular sentence, term, and goal translations.
- Implemented attributed variables.
- First CLPQ/CLPR implementation.
- Added the posibility of linking external .so files.

- Changes in syntax to allow `P(X)` and `"string"||L`.
- Changed to be closer to ISO-Prolog.
- Implemented Prolog shell scripts.
- Implemented data predicates.

Version 0.1 (1997/2/13)
First fully integrated, standalone Ciao distribution. Based on integrating into an evolution of the &-Prolog engine/libraries/preprocessor [Her86,HG91] many functionalities from several previous independent development versions of Ciao [HC93,HC94,HCC95,Bue95,CLI95,HBdlBP95,HBC96,CHV96b,HBC99].

2 Getting started on Un*x-like machines

Author(s): Manuel Hermenegildo.

This part guides you through some very basic first steps with Ciao on a Un*x-like system. It assumes that Ciao is already installed correctly on your Un*x system. If this is not the case, then follow the instructions in Chapter 231 [Installing Ciao from the source distribution], page 1101 first.

We start with by describing the basics of using Ciao from a normal command shell such as `sh/bash`, `csh/tcsh`, etc. We strongly recommend reading also Section 2.4 [An introduction to the Ciao emacs environment (Un*x)], page 24 for the basics on using Ciao under `emacs`, which is a much simpler and much more powerful way of developing Ciao programs, and has the advantage of offering an almost identical environment under Un*x and Windows.

2.1 Testing your Ciao Un*x installation

It is a good idea to start by performing some tests to check that Ciao is installed correctly on your system (these are the same tests that you are instructed to do during installation, so you can obviously skip them if you have done them already at that time). If any of these tests do not succeed either your environment variables are not set properly (see Section 2.2 [Un*x user setup], page 21 for how to fix this):

- Typing `ciao` (or `ciaosh`) should start the typical Prolog-style top-level shell.
- In the top-level shell, Ciao library modules should load correctly. Type for example `use_module(library(dec10_io))` –you should get back a prompt with no errors reported.
- To exit the top level shell, type `halt.` as usual, or ⟨^D⟩.
- Typing `ciaoc` should produce the help message from the Ciao standalone compiler.
- Typing `ciao-shell` should produce a message saying that no code was found. This is a Ciao application which can be used to write scripts written in Ciao, i.e., files which do not need any explicit compilation to be run.

Also, the following documentation-related actions should work:

- If the `info` program is installed, typing `info` should produce a list of manuals which *should include Ciao manual(s) in a separate area* (you may need to log out and back in so that your shell variables are reinitialized for this to work).
- Opening with a WWW browser (e.g., `netscape`) the directory or `URL` corresponding to the `DOCDIR` setting should show a series of Ciao-related manuals. Note that *style sheets* should be activated for correct formatting of the manual.
- Typing `man ciao` should produce a man page with some very basic general information on Ciao (and pointing to the on-line manuals).
- The `DOCDIR` directory should contain the manual also in the other formats such as `postscript` or `pdf` which specially useful for printing. See Section 2.3.7 [Printing manuals (Un*x)], page 24 for instructions.

2.2 Un*x user setup

If the tests above have succeeded, the system is probably installed correctly and your environment variables have been set already. In that case you can skip to the next section.

Otherwise, if you have not already done so, make the following modifications in your startup scripts, so that these files are used (`<LIBROOT>` must be replaced with the appropriate value, i.e., where the Ciao library is installed):

- For users a *csh-compatible shell* (`csh`, `tcsh`, ...), add to `~/.cshrc`:

```
if ( -e <v>libroot</v>/ciao/DOTcshrc ) then
   source <v>libroot</v>/ciao/DOTcshrc
endif
```

Note: while this is recognized by the terminal shell, and therefore by the text-mode Emacs which comes with Mac OS X, the Aqua native Emacs 21 does not recognize that initialization. It is thus necessary, at this moment, to set manually the Ciao shell (ciaosh) and Ciao library location by hand. This can be done from the Ciao menu within Emacs after a Ciao file has been loaded. We believe that the reason is that Mac OS X does not actually consult the per-user initialization files on startup. It should also be possible to put the right initializations in the .emacs file using the `setenv` function of Emacs-lisp, as in

```
(setenv "CIAOLIB" "<v>libroot</v>/ciao")
```

The same can be done for the rest of the variables initialized in `<v>libroot</v>/ciao/DOTcshrc`

- For users of an *sh-compatible shell* (`sh`, `bash`, ...), the installer will add to `~/.bashrc` the next lines:

```
if [ -f <v>libroot</v>/ciao/DOTprofile ]; then
   . <v>libroot</v>/ciao/DOTprofile
fi
```

This will set up things so that the Ciao executables are found and you can access the Ciao system manuals using the `info` command. Note that, depending on your shell, *you may have to log out and back in* for the changes to take effect.

- Also, if you use `emacs` (highly recommended) the install will add the next line to your `~/.emacs` file:

```
(load-file "<v>libroot</v>/ciao/ciao-mode-init.el")
(if (file-exists-p "<v>libroot</v>/ciao/ciao-mode-init.el")
  (load-file "<v>libroot</v>/ciao/ciao-mode-init.el")
)
```

If after following these steps things do not work properly, then the installation was probably not completed properly and you may want to try reinstalling the system.

2.3 Using Ciao from a Un*x command shell

2.3.1 Starting/exiting the top-level shell (Un*x)

The basic methods for starting/exiting the top-level shell have been discussed above. If upon typing `ciao` you get a "command not found" error or you get a longer message from Ciao before starting, it means that either Ciao was not installed correctly or you environment variables are not set up properly. Follow the instructions on the message printed by Ciao or refer to the installation instructions regarding user-setup for details.

2.3.2 Getting help (Un*x)

The basic methods for accessing the manual on-line have also been discussed above. Use the table of contents and the indices of *predicates*, *libraries*, *concepts*, etc. to find what you are looking for. Context-sensitive help is available within the `emacs` environment (see below).

2.3.3 Compiling and running programs (Un*x)

Once the shell is started, you can compile and execute modules inside the interactive top-level shell in the standard way. E.g., type `use_module(`*file*`).`, `use_module(library(`*file*`)).` for

library modules, `ensure_loaded(`*file*`).` for files which are not modules, and `use_package(`*file*`).` for library packages (these are syntactic/semantic packages that extend the Ciao language in many different ways). Note that the use of `compile/1` and `consult/1` is discouraged in Ciao.

For example, you may want to type `use_package(iso)` to ensure Ciao has loaded all the ISO builtins (whether this is done by default or not depends on your `.ciaorc` file). Do not worry about any "module already in executable" messages –these are normal and simply mean that a certain module is already pre-loaded in the top-level shell. At this point, typing `write(hello).` should work.

Note that some predicates that may be built-ins in typical Prolog implementations are available through libraries in Ciao. This facilitates making small executables.

To change the working directory to, say, the `examples` directory in the Ciao root directory, first do:

```
?- use_module(library(system)).
```

(loading the `system` library makes a number of system-related predicates such as `cd/1` accessible) and then:

```
?- cd('$/examples').
```

(in Ciao the sequence `$/` *at the beginning of a path name* is replaced by the path of the Ciao root directory).

For more information see Chapter 5 [The interactive top-level shell], page 41.

2.3.4 Generating executables (Un*x)

Executables can be generated from the top-level shell (using `make_exec/2`) or using the standalone compiler (`ciaoc`). To be able to make an executable, the file should define the predicate `main/1` (or `main/0`), which will be called upon startup (see the corresponding manual section for details). In its simplest use, given a top-level *foo*`.pl` file for an application, the compilation process produces an executable `foo`, automatically detecting which other files used by `foo.pl` need recompilation.

For example, within the `examples` directory, you can type:

```
?- make_exec(hw,_).
```

which should produce an executable. Typing `hw` in a shell (or double-clicking on the icon from a graphical window) should execute it.

For more information see Chapter 5 [The interactive top-level shell], page 41 and Chapter 4 [The stand-alone command-line compiler], page 33.

2.3.5 Running Ciao scripts (Un*x)

Ciao allows writing scripts. These are files containing Ciao source but which get executed without having to explicitly compile them (in the same way as, e.g., `.bat` files or programs in scripting languages). As an example, you can run the file `hw` in the `examples` directory of the Ciao distribution and look at the source with an editor. You can try changing the `Hello world` message and running the program again (no need to recompile!).

As you can see, the file should define the predicate `main/1` (not `main/0`), which will be called upon startup. The two header lines are necessary in Un*x in. In Windows you can leave them in or you can take them out, but you need to rename the script to `hw.pls`. Leaving the lines in has the advantage that the script will also work in Un*x without any change.

For more information see Chapter 8 [The script interpreter], page 63.

2.3.6 The Ciao initialization file (Un*x)

The Ciao toplevel can be made to execute upon startup a number of commands (such as, e.g., loading certain files or setting certain Ciao flags) contained in an initialization file. This file should be called `.ciaorc` and placed in your *home* directory (e.g., `~`, the same in which the `.emacs` file is put). You may need to set the environment variable `HOME` to the path of this directory for the Ciao toplevel shell to be able to locate this file on startup.

2.3.7 Printing manuals (Un*x)

As mentioned before, the manual is available in several formats in the `reference` directory within the `doc` directory in the Ciao distribution, including `postscript` or `pdf`, which are specially useful for printing. These files are also available in the `DOCDIR` directory specified during installation. Printing can be done using an application such as `ghostview` (freely available from `http://www.cs.wisc.edu/~ghost/index.html`) or `acrobat reader` (`http://www.adobe.com`, only `pdf`).

2.4 An introduction to the Ciao emacs environment (Un*x)

While it is easy to use Ciao with any editor of your choice, using it within the `emacs` editor/program development system is highly recommended: Ciao includes an `emacs` *mode* which provides a very complete *application development environment* which greatly simplifies many program development tasks. See Chapter 10 [Using Ciao inside GNU emacs], page 67 for details on the capabilities of `ciao/ emacs` combination.

If the (freely available) `emacs` editor/environment is not installed in your system, we highly recommend that you also install it at this point (there are instructions for where to find `emacs` and how to install it in the Ciao installation instructions). After having done this you can try for example the following things:

- A few basic things:
  - Typing ⟨^H⟩ ⟨i⟩ (or in the menus `Help->Manuals->Browse Manuals with Info`) should open a list of manuals in info format in which the Ciao manual(s) should appear.
  - When opening a Ciao file, i.e., a file with `.pl` or `.pls` ending, using ⟨^X⟩⟨^F⟩`filename` (or using the menus) the code should appear highlighted according to syntax (e.g., comments in red), and `Ciao/Prolog` menus should appear in the menu bar on top of the `emacs` window.
  - Loading the file using the `Ciao/Prolog` menu (or typing ⟨^C⟩ ⟨l⟩) should start in another emacs buffer the Ciao toplevel shell and load the file. You should now be able to switch the the toplevel shell and make queries from within `emacs`.

  Note: when using `emacs` it is *very convenient* to swap the locations of the (normally not very useful) ⟨Caps Lock⟩ key and the (very useful in `emacs`) ⟨Ctrl⟩ key on the keyboard. How to do this is explained in the `emacs` frequently asked questions FAQs (see the `emacs` download instructions for their location).

  (if these things do not work the system or emacs may not be installed properly).
- You can go to the location of most of the errors that may be reported during compilation by typing ⟨^C⟩ ⟨`⟩.
- You can also, e.g., create executables from the `Ciao/Prolog` menu, as well as compile individual files, or generate active modules.
- Loading a file for source-level debugging using the `Ciao/Prolog` menu (or typing ⟨^C⟩ ⟨d⟩) and then issuing a query should start the source-level debugger and move a marker on the code in a window while execution is stepped through in the window running the Ciao top level.

- You can add the lines needed in Un*x for turning any file defining `main/1` into a script from the Ciao/Prolog menu or by typing ⟨^C⟩ ⟨I⟩ ⟨S⟩.
- You can also work with the preprocessor and auto-documenter directly from emacs: see their manuals or browse through the corresponding menus that appear when editing `.pl` files.

We encourage you once more to read Chapter 10 [Using Ciao inside GNU emacs], page 67 to discover the many other functionalities of this environment.

2.5 Keeping up to date (Un*x)

You may want to read Chapter 233 [Beyond installation], page 1115 for instructions on how to sign up on the Ciao user's mailing list, receive announcements regarding new versions, download new versions, report bugs, etc.

3 Getting started on Windows machines

Author(s): Manuel Hermenegildo.

This part guides you through some very basic first steps with Ciao on an MSWindows ("Win32") system. It assumes that Ciao is already installed correctly on your Windows system. If this is not the case, then follow the instructions in Chapter 232 [Installing Ciao from a Win32 binary distribution], page 1111 (or Chapter 231 [Installing Ciao from the source distribution], page 1101) first.

We start with by describing the basics of using Ciao from the Windows explorer and/or a DOS command shell. We strongly recommend reading also Section 3.3 [An introduction to the Ciao emacs environment (Win32)], page 29 for the basics on using Ciao under `emacs`, which is a much simpler and much more powerful way of developing Ciao programs, and has the advantage of offering an almost identical environment under Windows and Un*x.

3.1 Testing your Ciao Win32 installation

It is a good idea to start by performing some tests to check that Ciao is installed correctly on your system (these are the same tests that you are instructed to do during installation, so you can obviously skip them if you have done them already at that time):

- Ciao-related file types (`.pl` source files, `.cpx` executables, `.itf,.po,.asr` interface files, `.pls` scripts, etc.) should have specific icons associated with them (you can look at the files in the folders in the Ciao distribution to check).
- Double-clicking on the shortcut to `ciaosh(.cpx)` on the desktop should start the typical Prolog-style top-level shell in a window. If this shortcut has not been created on the desktop, then double-clicking on the `ciaosh(.cpx)` icon inside the `shell` folder within the Ciao source folder should have the same effect.
- In the top-level shell, Ciao library modules should load correctly. Type for example `use_module(library(dec10_io)).` at the Ciao top-level prompt –you should get back a prompt with no errors reported.
- To exit the top level shell, type `halt.` as usual, or ⟨^D⟩.

Also, the following documentation-related actions should work:

- Double-clicking on the shortcut to `ciao(.html)` which appears on the desktop should show the Ciao manual in your default WWW browser. If this shortcut has not been created you can double-click on the `ciao(.html)` file in the `doc\reference\ciao_html` folder inside the Ciao source folder. Make sure you configure your browser to use *style sheets* for correct formatting of the manual (note, however, that some older versions of Explorer did not support style sheets well and will give better results turning them off).
- The `doc\reference` folder contains the manual also in the other formats present in the distribution, such as `info` (very convenient for users of the `emacs` editor/program development system) and `postscript` or `pdf`, which are specially useful for printing. See Section 3.2.7 [Printing manuals (Win32)], page 29 for instructions.

3.2 Using Ciao from the Windows explorer and command shell

3.2.1 Starting/exiting the top-level shell (Win32)

The basic methods for starting/exiting the top-level shell have been discussed above. The installation script also leaves a `ciaosh(.bat)` file inside the `shell` folder of the Ciao distribution which can be used to start the top-level shell from the command line in Windows systems.

3.2.2 Getting help (Win32)

The basic methods for accessing the manual on-line have also been discussed above. Use the table of contents and the indices of *predicates*, *libraries*, *concepts*, etc. to find what you are looking for. Context-sensitive help is available within the `emacs` environment (see below).

3.2.3 Compiling and running programs (Win32)

Once the shell is started, you can compile and execute Ciao modules inside the interactive toplevel shell in the standard way. E.g., type `use_module(`*file*`).`, `use_module(library(`*file*`)).` for library modules, `ensure_loaded(`*file*`).` for files which are not modules, and `use_package(`*file*`).` for library packages (these are syntactic/semantic packages that extend the Ciao language in many different ways). Note that the use of `compile/1` and `consult/1` is discouraged in Ciao.

For example, you may want to type `use_package(iso)` to ensure Ciao has loaded all the ISO builtins (whether this is done by default or not depends on your `.ciaorc` file). Do not worry about any "module already in executable" messages –these are normal and simply mean that a certain module is already pre-loaded in the toplevel shell. At this point, typing `write(hello).` should work.

Note that some predicates that may be built-ins in typical Prolog implementations are available through libraries in Ciao. This facilitates making small executables.

To change the working directory to, say, the `examples` directory in the Ciao source directory, first do:

```
?- use_module(library(system)).
```

(loading the `system` library makes a number of system-related predicates such as `cd/1` accessible) and then:

```
?- cd('$/examples').
```

(in Ciao the sequence `$/` *at the beginning of a path name* is replaced by the path of the Ciao root directory).

For more information see Chapter 5 [The interactive top-level shell], page 41.

3.2.4 Generating executables (Win32)

Executables can be generated from the toplevel shell (using `make_exec/2`) or using the standalone compiler (`ciaoc(.cpx)`, located in the `ciaoc` folder). To be able to make an executable, the file should define the predicate `main/1` (or `main/0`), which will be called upon startup (see the corresponding manual section for details).

For example, within the `examples` directory, you can type:

```
?- make_exec(hw,_).
```

which should produce an executable. Double-clicking on this executable should execute it.

Another way of creating Ciao executables from source files is by right-clicking on `.pl` files and choosing "make executable". This uses the standalone compiler (this has the disadvantage, however, that it is sometimes difficult to see the error messages).

For more information see Chapter 5 [The interactive top-level shell], page 41 and Chapter 4 [The stand-alone command-line compiler], page 33.

3.2.5 Running Ciao scripts (Win32)

Double-clicking on files ending in `.pls`, *Ciao scripts*, will also execute them. These are files containing Ciao source but which get executed without having to explicitly compile them (in the same way as, e.g., `.bat` files or programs in scripting languages). As an example, you can double-click on the file `hw.pls` in the `examples` folder and look at the source with an editor. You can try changing the `Hello world` message and double-clicking again (no need to recompile!).

As you can see, the file should define the predicate `main/1` (not `main/0`), which will be called upon startup. The two header lines are only necessary in Un*x. In Windows you can leave them in or you can take them out, but leaving them in has the advantage that the script will also work in Un*x without any change.

For more information see Chapter 8 [The script interpreter], page 63.

3.2.6 The Ciao initialization file (Win32)

The Ciao toplevel can be made to execute upon startup a number of commands (such as, e.g., loading certain files or setting certain Ciao flags) contained in an initialization file. This file should be called `.ciaorc` and placed in your *home* folder (e.g., the same in which the `.emacs` file is put). You may need to set the environment variable `HOME` to the path of this folder for the Ciao toplevel shell to be able to locate this file on startup.

3.2.7 Printing manuals (Win32)

As mentioned before, the manual is available in several formats in the `reference` folder within Ciao's `doc` folder, including `postscript` or `pdf`, which are specially useful for printing. This can be done using an application such as `ghostview` (freely available from `http://www.cs.wisc.edu/~ghost/index.html`) or `acrobat reader` (`http://www.adobe.com`, only `pdf`).

3.3 An introduction to the Ciao emacs environment (Win32)

While it is easy to use Ciao with any editor of your choice, using it within the `emacs` editor/program development system is highly recommended: Ciao includes an `emacs` *mode* which provides a very complete *application development environment* which greatly simplifies many program development tasks. See Chapter 10 [Using Ciao inside GNU emacs], page 67 for details on the capabilities of `ciao/ emacs` combination.

If the (freely available) `emacs` editor/environment is not installed in your system, we highly recommend that you also install it at this point (there are instructions for where to find `emacs` and how to install it in the Ciao installation instructions). After having done this you can try for example the following things:

- A few basic things:
  - Typing ⟨^H⟩ ⓘ (or in the menus `Help->Manuals->Browse Manuals with Info`) should open a list of manuals in info format in which the Ciao manual(s) should appear.
  - When opening a Ciao file, i.e., a file with `.pl` or `.pls` ending, using ⟨^X⟩⟨^F⟩`filename` (or using the menus) the code should appear highlighted according to syntax (e.g., comments in red), and `Ciao/Prolog` menus should appear in the menu bar on top of the `emacs` window.
  - Loading the file using the `Ciao/Prolog` menu (or typing ⟨^C⟩ ⓛ) should start in another emacs buffer the Ciao toplevel shell and load the file. You should now be able to switch the the toplevel shell and make queries from within `emacs`.

Note: when using `emacs` it is *very convenient* to swap the locations of the (normally not very useful) ⟨Caps Lock⟩ key and the (very useful in `emacs`) ⟨Ctrl⟩ key on the keyboard. How to do this is explained in the `emacs` frequently asked questions FAQs (see the `emacs` download instructions for their location).

(if these things do not work the system or emacs may not be installed properly).

- You can go to the location of most of the errors that may be reported during compilation by typing ⟨^C⟩ ⟨`⟩.
- You can also, e.g., create executables from the `Ciao/Prolog` menu, as well as compile individual files, or generate active modules.
- Loading a file for source-level debugging using the `Ciao/Prolog` menu (or typing ⟨^C⟩ ⟨d⟩) and then issuing a query should start the source-level debugger and move a marker on the code in a window while execution is stepped through in the window running the Ciao top level.
- You can add the lines needed in Un*x for turning any file defining `main/1` into a script from the Ciao/Prolog menu or by typing ⟨^C⟩ ⟨I⟩ ⟨S⟩.
- You can also work with the preprocessor and auto-documenter directly from emacs: see their manuals or browse through the corresponding menus that appear when editing `.pl` files.

We encourage you once more to read Chapter 10 [Using Ciao inside GNU emacs], page 67 to discover the many other functionalities of this environment.

3.4 Keeping up to date (Win32)

You may want to read Chapter 233 [Beyond installation], page 1115 for instructions on how to sign up on the Ciao user's mailing list, receive announcements regarding new versions, download new versions, report bugs, etc.

PART I - The program development environment

Author(s): The CLIP Group.

This part documents the components of the basic Ciao program development environment. They include:

ciaoc: the standalone compiler, which creates executables without having to enter the interactive top-level.

ciaosh: (also invoked simply as `ciao`) is an interactive top-level shell, similar to the one found on most Prolog systems (with some enhancements).

debugger.pl: a Byrd box-type debugger, similar to the one found on most Prolog systems (also with some enhancements, such as source-level debugging). This is not a standalone application, but is rather included in `ciaosh`, as is done in other systems supporting the Prolog language. However, it is also *embeddable*, in the sense that it can be included as a library in executables, and activated dynamically and conditionally while such executables are running.

ciao-shell: an interpreter/compiler for *Ciao scripts* (i.e., files containing Ciao code which run without needing explicit compilation).

ciao.el: a *complete program development enviroment*, based on GNU emacs, with syntax coloring, direct access to all the tools described above (as well as the preprocessor and the documenter), atomatic location of errors, source-level debugging, context-sensitive access to on-line help/manuals, etc. The use of this environment is *very highly recommended*!

The Ciao program development environment also includes `ciaopp`, the preprocessor, and `lpdoc`, the documentation generator, which are described in separate manuals.

4 The stand-alone command-line compiler

Author(s): Daniel Cabeza, Edison Mera, The CLIP Group.

`ciaoc` [CH00b] is the Ciao stand-alone command-line compiler. `ciaoc` can be used to create executables or to compile individual files to object code (to be later linked with other files). `ciaoc` is specially useful when working from the command line. Also, it can be called to compile Ciao programs from other tools such as, e.g., shell scripts, `Makefile`s, or project files. All the capabilities of `ciaoc` are also available from the interactive top-level shell, which uses the ciaoc modules as its components.

4.1 Introduction to building executables

An *executable* can be built from a single file or from a collection of inter-related files. In the case of only one file, this file must define the predicate `main/0` or `main/1`. This predicate is the one which will be called when the executable is started. As an example, consider the following file, called `hello.pl`:

```
main :-
     write('Hello world'),
     nl.
```

To compile it from the command line using the `ciaoc` standalone compiler it suffices to type "`ciaoc hello`" (in Win32 you may have to put the complete path to the `ciaoc` folder of the Ciao distribution, where the installation process leaves a `ciaoc.bat` file):

```
/herme@clip:/tmp
[60]> ciaoc hello

/herme@clip:/tmp
[61]>
```

This produces an executable called `hello` in Un*x-like systems and `hello.cpx` under Win32 systems. This executable can then be run in Win32 by double-clicking on it and on Un*x systems by simply typing its name (see for Section 4.3 [Running executables from the command line], page 34 for how to run executables from the command line in Win32):

```
/herme@clip:/tmp
[61]> hello
Hello world
```

If the application is composed of several files the process is identical. Assume `hello.pl` is now:

```
:- use_module(aux, [p/1]).

main :-
     p(X),
     write(X),
     nl.
```

where the file `aux.pl` contains:

```
:- module(aux,[p/1]).

p('Hello world').
```

This can again be compiled using the `ciaoc` standalone compiler as before:

```
/herme@clip:/tmp
[60]> ciaoc hello

/herme@clip:/tmp
[61]> hello
Hello world
```

The invocation of `ciaoc hello` compiles the file `hello.pl` and all connected files that may need recompilation – in this case the file `aux.pl`. Also, if any library files used had not been compiled previously they would be compiled at this point (See Section 4.6 [Intermediate files in the compilation process], page 37). Also, if, say, `hello.pl` is changed and recompiled, the object code resulting from the previous compilation of `aux.pl` will be reused. This is all done without any need for `Makefile`s, and considerably accelerates the development process for large applications. This process can be observed by selecting the `-v` option when invoking `ciaoc` (which is equivalent to setting the `verbose_compilation` Prolog flag to `on` in the top-level interpreter).

If `main/1` is defined instead of `main/0` then when the executable is started the argument of `main/1` will be instantiated to a list of atoms, each one of them corresponding to a command line option. Consider the file `say.pl`:

```
main(Argv) :-
     write_list(Argv), nl.

write_list([]).
write_list([Arg|Args]) :-
     write(Arg),
     write(' '),
     write_list(Args).
```

Compiling this program and running it results in the following output:

```
/herme@clip:/tmp
[91]> ciaoc say

/herme@clip:/tmp
[91]> say hello dolly
hello dolly
```

The name of the generated executable can be controlled with the `-o` option (See Section 4.7 [Usage (ciaoc)], page 38).

4.2 Paths used by the compiler during compilation

The compiler will look for files mentioned in commands such as `use_module/1` or `ensure_loaded/1` in the current directory. Other paths can be added by including them in a file whose name is given to `ciaoc` using the `-u` option. This file should contain facts of the predicates `file_search_path/2` and `library_directory/1` (see the documentation for these predicates and also Chapter 9 [Customizing library paths and path aliases], page 65 for details).

4.3 Running executables from the command line

As mentioned before, what the `ciaoc` compiler generates and how it is started varies somewhat from OS to OS. In general, the product of compiling an application with `ciaoc` is a file that contains the bytecode (the product of the compilation) and invokes the Ciao engine on it.

- Un Un*x this is a *script* (see the first lines of the file) which invokes the ciao engine on this file. To run the generated executable from a Un*x shell, or from the `bash` shell that comes with the Cygwin libraries (see Section 231.6 [Installation and compilation under Windows], page 1107) it suffices to type its name at the shell command line, as in the examples above.
- In a Win32 system, the compiler produces a similar file with a `.cpx` ending. The Ciao installation process typically makes sure that the Windows registry contains the right entries so that this executable will run upon double-cliking on it.

  In you want to run the executable from the command line an additional `.bat` file is typically needed. To help in doing this, the Win32 installation process creates a `.bat` skeleton file called `bat_skel` in the `Win32` folder of the distribution) which allows running Ciao executables from the command line. If you want to run a Ciao executable `file.cpx` from the command line, you normally copy the skeleton file to the folder were the executable is and rename it to `file.bat`, then change its contents as explained in a comment inside the file itself.

  Note that this `.bat` file is usually not necessary in NT, as its command shell understands file extension associations. I.e., in windows NT it is possible to run the `file.cpx` executable directly. Due to limitations of `.bat` files in Windows 95/98, in those OSs no more than 9 command line arguments can be passed to the executable (in NT there is no such restriction).

  Finally, in a system in which Cygnus Win32 is installed executables can also be used directly from the `bash` shell command line, without any associated `.bat` files, by simply typing their name at the `bash` shell command line, in the same way as in Un*x. This only requires that the `bash` shell which comes with Cygnus Win32 be installed and accessible: simply, make sure that `/bin/sh.exe` exists.

Except for a couple of header lines, the contents of executables are almost identical under different OSs (except for self-contained ones). The bytecode they contain is architecture-independent. In fact, it is possible to create an executable under Un*x and run it on Windows or viceversa, by making only minor modifications (e.g., creating the `.bat` file and/or setting environment variables or editing the start of the file to point to the correct engine location).

4.4 Types of executables generated

While the default options used by `ciaoc` are sufficient for normal use, by selecting other options `ciaoc` can generate several different types of executables, which offer interesting tradeoffs among size of the generated executable, portability, and startup time [CH00b]:

Dynamic executables:

`ciaoc` produces by default *dynamic* executables. In this case the executable produced is a platform-independent file which includes in compiled form all the user defined files. On the other hand, any system libraries used by the application are loaded dynamically at startup. More precisely, any files that appear as `library(...)` in `use_module/1` and `ensure_loaded/1` declarations will not be included explicitly in the executable and will instead be loaded dynamically. Is is also possible to mark other path aliases (see the documentation for `file_search_path/2`) for dynamic loading by using the `-d` option. Files accessed through such aliases will also be loaded dynamically.

Dynamic loading allows making smaller executables. Such executables may be used directly in the same machine in which they were compiled, since suitable paths to the location of the libraries will be included as default in the executable by `ciaoc` during compilation.

The executable can also be used in another machine, even if the architecture and OS are different. The requirement is that the Ciao libraries (which will also include the appropriate Ciao engine for that architecture and OS) be installed in the target

machine, and that the `CIAOLIB` and `CIAOENGINE` environment variables are set appropriately for the executable to be able to find them (see Section 4.5 [Environment variables used by Ciao executables], page 37). How to do this differs slightly from OS to OS.

Static executables:
: Selecting the `-s` option `ciaoc` produces a *static* executable. In this case the executable produced (again a platform-independent file) will include in it all the auxiliary files and any system libraries needed by the application. Thus, such an executable is almost complete, needing in order to run only the Ciao engine, which is platform-specific.[1] Again, if the executable is run in the same machine in which it was compiled then the engine is found automatically. If the executable is moved to another machine, the executable only needs access to a suitable engine (which can be done by setting the `CIAOENGINE` environment variable to point to this engine).

  This type of compilation produces larger executables, but has the advantage that these executables can be installed and run in a different machine, with different architecture and OS, even if Ciao is not installed on that machine. To install (or distribute) such an executable, one only needs to copy the executable file itself and the appropriate engine for the target platform (See Chapter 231 [Installing Ciao from the source distribution], page 1101 or Chapter 232 [Installing Ciao from a Win32 binary distribution], page 1111 and Section 231.5 [Multiarchitecture support], page 1106), and to set things so that the executable can find the engine.[2]

Dynamic executables, with lazy loading:
: Selecting the `-l` option is very similar to the case of dynamic executables above, except that the code in the library modules is not loaded when the program is started but rather it is done during execution, the first time a predicate defined in that file is called. This is advantageous if a large application is composed of many parts but is such that typically only some of the parts are used in each invocation. The Ciao preprocessor, `ciaopp`, is a good example of this: it has many capabilitites but typically only some of them are used in a given session. An executable with lazy load has the advantage that it starts fast, loading a minimal functionality on startup, and then loads the different modules automatically as needed.

Self-contained executables:
: *Self-contained* executables are static executables (i.e., this option also implies *static* compilation) which include a Ciao engine along with the bytecode, so they do not depend on an external one for their execution. This is useful to create executables which run even if the machine where the program is to be executed does not have a Ciao engine installed and/or libraries. The disadvantage is that such execuatbles are platform-dependent (as well as larger than those that simply use an external library). This type of compilation is selected with the `-S` option. Cross-compilation is also possible with the `-SS` option, so you can specify the target OS and architecture (e.g.

[1] Currently there is an exception to this related to libraries which are written in languages other than Prolog, as, e.g., C. C files are currently always compiled to dynamically loadable object files (`.so` files), and they thus need to be included manually in a distribution of an application. This will be automated in upcoming versions of the Ciao system.

[2] It is also possible to produce real standalone executables, i.e., executables that do not need to have an engine around. However, this is not automated yet, although it is planned for an upcoming version of the compiler. In particular, the compiler can generate a `.c` file for each `.pl` file. Then all the `.c` files can be compiled together into a real executable (the engine is added one more element during link time) producing a complete executable for a given architecture. The downside of course is that such an executable will not be portable to other architectures without recompilation.

LINUXi86). To be able to use the latter option, it is necessary to have installed a ciaoengine for the target machine in the Ciao library (this requires compiling the engine in that OS/architecture and installing it, so that it is available in the library).

Compressed executables:
: In *compressed* executables the bytecode is compressed. This allows producing smaller executables, at the cost of a slightly slower startup time. This is selected with the `-z` option. You can also produce compressed libraries if you use `-zl` along with the `-c` option. If you select `-zl` while generating an executable, any library which is compiled to accomplish this will be also compressed.

Active modules:
: The compiler can also compile (via the `-a` option) a given file into an *active module* (see Chapter 110 [Active modules (high-level distributed execution)], page 585 for a description of this).

4.5 Environment variables used by Ciao executables

The executables generated by the Ciao compiler (including the ciao development tools themselves) locate automatically where the Ciao engine and libraries have been installed, since those paths are stored as defaults in the engine and compiler at installation time. Thus, there is no need for setting any environment variables in order to *run* Ciao executables (on a single architecture – see Section 231.5 [Multiarchitecture support], page 1106 for running on multiple architectures).

However, the default paths can be overridden by using the environment variables `CIAOENGINE` and `CIAOLIB`. The first one will tell the Ciao executables where to look for an engine, and the second will tell them where to look for the libraries. Thus, it is possible to actually use the Ciao system without installing it by setting these variables to the following values:

- `CIAOENGINE`: `$(CIAOSRC)/build/bin/$(CIAOARCH)/ciaoengine`
- `CIAOLIB`: `$(CIAOSRC)`

where `$(CIAOARCH)` is the string echoed by the command `CIAOSRC/etc/ciao_get_arch` (or `BINROOT/ciao_get_arch`, after installation).

This allows using alternate engines or libraries, which can be very useful for system development and experimentation.

4.6 Intermediate files in the compilation process

Compiling an individual source (i.e., `.pl`) file produces a `.itf` file and a `.po` file. The `.itf` file contains information of the *modular interface* of the file, such as information on exported and imported predicates and on the other modules used by this module. This information is used to know if a given file should be recompiled at a given point in time and also to be able to detect more errors statically including undefined predicates, mismatches on predicate charaterictics across modules, etc. The `.po` file contains the platform-independent object code for a file, ready for linking (statically or dynamically).

It is also possible to use `ciaoc` to explicitly generate the `.po` file for one or more `.pl` files by using the `-c` option.

If you want to view the wam instructions of one or more `.pl` files you can use the `-w` option. That will generate a `.wam` file with such instructions in a pretty format per each `.pl` file.

4.7 Usage (ciaoc)

The following provides details on the different command line options available when invoking `ciaoc`:

```
ciaoc <MiscOpts> <ExecOpts> [-o <execname>] <file> ...

  Make an executable from the listed files.  If there is
  more than one file, they must be non-module, and the
  first one must include the main predicate.  The -o
  option allows generating an arbitrary executable name.

ciaoc <MiscOpts> <ExecOpts> -a <publishmod> <module>

  Make an active module executable from <module> with
  address publish module <publishmod>.

ciaoc <MiscOpts> -c  <file> ...

  Compile listed files (make .po objects).

ciaoc <MiscOpts> -w  <file> ...

  Generate WAM code of listed files (in .wam files).

<MiscOpts> can be: [-v] [-ri] [-u <file>]  [-rc] [-op <suffix>] [-L <LibDir>]

<ExecOpts> can be: [-s|-S|-SS <target>|-z|-zl|-e|-l|(-ll <module>)*]
                   (-d <alias>)* [-x]

default extension for files is '.pl'

-h, --help
Show this help.
-u use <file> for compilation, often used to include LibDir paths, etc.
-op use <suffix> as the suffix for optimized (or otherwise tuned) code
-L look for libraries also in the <LibDir> directory
-c Compile listed files (make .po objects)
-w Generate WAM code of listed files (in .wam files).
-S
make standalone executable for the current OS and architecture, implies -s
-SS make standalone executable for <target> OS and architecture
valid <target> values may be: LINUXi86, SolarisSparc..., implies -s
-ll force <module> to be loaded lazily,  implies -l
-ac All the modules will be compiled using <Packages>
-acm <Modules> will be compiled using <Packages>
-d files using this path alias are dynamic (default: library)
-o Make an executable from the listed files.
-a Make an active module
-v, --verbose-compilation
verbose mode
```

```
-ri, --itf-format-r
Generate human readable .itf files
-x, --check-libraries
extended recompilation: only useful for Ciao standard library developers
-s, --executables-static
make a static executable (otherwise dynamic files are not included)
-z, --compress-exec
Generate executables with compressed bytecode
-zl, --compress-lib
generate libraries with compressed bytecode - any library (re)compiled as
consequence of normal executable compilation will also be affected
-l, --executables-lazyload
Idem with lazy load of dynamic files (except insecure cases)
-np, --use-compile-packages-no
Do not use compile packages
-na, --read-assertions-no
Do not read the assertions in the code
-rc, --runtime-checks
Generate code with runtime checks, requires to read assertions
--rtchecks-trust-no
Disable rtchecks for trust assertions
--rtchecks-entry-no
Disable rtchecks for entry assertions
--rtchecks-exit-no
Disable rtchecks for exit assertions
--rtchecks-test
enable rtchecks for test assertions. Used for debugging
purposes, but is better to use the unittest library
--rtchecks-level-exports
Use rtchecks only for external calls of the exported predicates
--rtchecks-inline
Expand library predicates inline as far as possible
--rtchecks-asrloc-no
Do not use assertion locators in the error messages
--rtchecks-predloc-no
Do not use predicate locators in the error messages
--rtchecks-namefmt-short
Show the name of predicates and properties in a reduced format
--rtchecks-callloc-no
Do not show the stack of predicates that caused the failure
--rtchecks-callloc-literal
Show the stack of predicates that caused the failure. Instrument it
in the literal. This mode provides more information, because reports
also the literal in the body of the predicate
--unused-pred-warnings
Show warnings about unused predicates.  Note that a predicate is
being used if it is exported, it appears in clause body of a
predicate being used, in a multifile predicate, in a predicate
used in :- initialization(...) or :- on_abort(...)
declarations, or if it is the meta-argument of a metapredicate.
```

4.8 Known bugs and planned improvements (`ciaoc`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.
- Also if appears in the body of an assertion referred to a predicate being used, but that is not implemented, because the assertion reader is not included in the compiler yet – EMM.

5 The interactive top-level shell

Author(s): Daniel Cabeza, The CLIP Group.

`ciaosh` is the Ciao interactive top-level shell. It provides the user with an interactive programming environment with tools for incrementally building programs, debugging programs by following their executions, and modifying parts of programs without having to start again from scratch. If available, it is strongly recommended to use it with the emacs interface provided, as it greatly simplifies the operation. This chapter documents general operation in the shell itself.

5.1 Shell invocation and startup

When invoked, the shell responds with a message of identification and the prompt `?-` as soon as it is ready to accept input, thus:

```
Ciao-Prolog X.Y #PP: Thu Mar 25 17:20:55 MET 1999
?-
```

When the shell is initialized it looks for a file `.ciaorc` in the HOME directory and makes an `include` of it, if it exists. This file is useful for including `use_module/1` declarations for the modules one wants to be loaded by default, changing prolog flags, etc. (Note that the `.ciaorc` file can only contain directives, not actual code; to load some code at startup put it in a separate file and load it using e.g. a `use_module/1` declaration.) If the initialization file does not exist, the default package `default_for_ciaosh` is included, to provide more or less what other prologs define by default. Thus, if you want to have available all builtins you had before adding the initialization file, you have to include `:- use_package(default_for_ciaosh)` in it. Two command-line options control the loading of the initialization file:

`-f` Fast start, do not load any initialization file.

`-l File` Look for initialization file `File` instead of `~/.ciaorc`. If it does not exist, include the default package.

5.2 Shell interaction

After the shell outputs the prompt, it is expecting either an internal command (see the following sections) or a *query* (a goal or sequence of goals). When typing in the input, which must be a valid prolog term, if the term does not end in the first line, subsequent lines are indented. For example:

```
?- X =
   f(a,
   b).

X = f(a,b) ?

yes
?-
```

The queries are executed by the shell as if they appeared in the user module. Thus, in addition to builtin predicates, predicates available to be executed directly are all predicates defined by loaded user files (files with no module declaration), and imported predicates from modules by the use of `use_module`.

The possible answers of the shell, after executing an internal command or query, are:

- If the execution failed (or produced an error), the answer is `no`.

- If the execution was successful and bindings where made (or constraints where imposed) on answer variables, then the shell outputs the values of answer variables, as a sequence of bindings (or constraints), and then prints a `?` as a prompt. At this point it is expecting an input line from the user. By entering a carriage-return (⟨RET⟩) or any line starting with `y`, the query terminates and the shell answer `yes`. Entering a '`,`' the shell enters a recursive level (see below). Finally, any other answer forces the system to backtrack and look for the next solution (answering as with the first solution).
- If the execution was successful, but no answer variable was bound or constrained, the answer is simply `yes`. This behavior can be changed by setting the prolog flag `prompt_alternatives_no_bindings` to `on`, so that if there are more solutions the user will be consulted as explained in the previous point (useful if the solutions produce side effects).

To allow using connection variables in queries without having to report their results, variables whose name starts with `_` are not considered in answers, the rest being the *answer variable*s. This example illustrates the previous points:

```
?- member(a, [b, c]).

no
?- member(a, [a, b]).

yes
?- member(X, [a|L]).

X = a ? ;

L = [X|_] ?

yes
?- atom_codes(ciao, _C), member(L, _C).

L = 99 ? ;

L = 105 ? ;

L = 97 ? ;

L = 111 ? ;

no
?-
```

5.3 Entering recursive (conjunctive) shell levels

As stated before, when the user answers with '`,`' after a solution is presented, the shell enters a *recursive level*, changing its prompt to *N* `?-` (where *N* is the recursion level) and keeping the bindings or constraints of the solution (this is inspired by the *LogIn* language developed by H. Ait-Kaci, P. Lincoln and Roger Nasr [AKNL86]). Thus, the following queries will be executed within that context, and all variables in the lower level solutions will be reported in subsequent solutions at this level. To exit a recursive level, input an ⟨EOF⟩ character or the command `up`. The last solution after entering the level is repeated, to allow asking for more solutions. Use command `top` to exit all recursive levels and return to the top level. Example interaction:

```
?- directory_files('.',_Fs), member(F,_Fs).
```

```
F = 'file_utils.po' ? ,

1 ?- file_property(F, mod_time(T)).

F = 'file_utils.po',
T = 923497679 ?

yes
1 ?- up.

F = 'file_utils.po' ? ;

F = 'file_utils.pl' ? ;

F = 'file_utils.itf' ? ,

1 ?- file_property(F, mod_time(T)).

F = 'file_utils.itf',
T = 923497679 ?

yes
1 ?- ^D
F = 'file_utils.itf' ?

yes
?-
```

5.4 Usage and interface (`toplevel_doc`)

- **Library usage:**
  The following predicates can be used at the top-level shell natively (but see also the commands available in Chapter 6 [The interactive debugger], page 49 which are also available within the top-level shell).
- **Exports:**
  - *Predicates:*
    `use_module/1`, `use_module/2`, `ensure_loaded/1`, `make_exec/2`, `include/1`, `use_package/1`, `consult/1`, `compile/1`, `./2`, `make_po/1`, `unload/1`, `set_debug_mode/1`, `set_nodebug_mode/1`, `make_actmod/2`, `force_lazy/1`, `undo_force_lazy/1`, `dynamic_search_path/1`, `multifile/1`.
- **Other modules used:**
  - *System library modules:*
    `toplevel/toplevel`, `libpaths`, `compiler/compiler`, `compiler/exemaker`, `compiler/c_itf`, `debugger/debugger`.

5.5 Documentation on exports (`toplevel_doc`)

use_module/1: PREDICATE

Usage: `use_module(Module)`

- *Description:* Load into the top-level the module defined in `Module`, importing all the predicates it exports.
- *The following properties should hold at call time:*

  `Module` is a source name. (streams_basic:sourcename/1)

use_module/2: PREDICATE

Usage: `use_module(Module,Imports)`

- *Description:* Load into the top-level the module defined in `Module`, importing the predicates in `Imports`.
- *The following properties should hold at call time:*

  `Module` is a source name. (streams_basic:sourcename/1)

  `Imports` is a list of `prednames`. (basic_props:list/2)

ensure_loaded/1: PREDICATE

Usage: `ensure_loaded(File)`

- *Description:* Load into the top-level the code residing in file (or files) `File`, which is user (i.e. non-module) code.
- *The following properties should hold at call time:*

  `File` is a source name or a list of source names. (toplevel_doc:sourcenames/1)

make_exec/2: PREDICATE

Usage: `make_exec(File,ExecName)`

- *Description:* Make a Ciao executable from file (or files) `File`, giving it name `ExecName`. If `ExecName` is a variable, the compiler will choose a default name for the executable and will bind the variable `ExecName` to that name. The name is chosen as follows: if the main prolog file has no `.pl` extension or we are in Windows, the executable will have extension `.cpx`; else the executable will be named as the main prolog file without extension.
- *The following properties should hold at call time:*

  `File` is a source name or a list of source names. (toplevel_doc:sourcenames/1)
- *The following properties hold upon exit:*

  `ExecName` is an atom. (basic_props:atm/1)

include/1: PREDICATE

Usage: `include(File)`

- *Description:* The contents of the file `File` are included in the top-level shell. For the moment, it only works with some directives, which are interpreted by the shell, or with normal clauses (which are asserted), if library(dynamic) is loaded beforehand.
- *The following properties should hold at call time:*

  `File` is a source name. (streams_basic:sourcename/1)

use_package/1: PREDICATE

Usage: `use_package(Package)`

- *Description:* Include the package or packages specified in `Package`. Most package contents can be handled in the top level, but there are currently still some limitations.
- *The following properties should hold at call time:*

  `Package` is a source name or a list of source names. (toplevel_doc:sourcenames/1)

consult/1: PREDICATE

Usage: `consult(File)`

- *Description:* Provided for backward compatibility. Similar to `ensure_loaded/1`, but ensuring each listed file is loaded in consult mode (see Chapter 6 [The interactive debugger], page 49).
- *The following properties should hold at call time:*

  `File` is a source name or a list of source names. (toplevel_doc:sourcenames/1)

compile/1: PREDICATE

Usage: `compile(File)`

- *Description:* Provided for backward compatibility. Similar to `ensure_loaded/1`, but ensuring each listed file is loaded in compile mode (see Chapter 6 [The interactive debugger], page 49).
- *The following properties should hold at call time:*

  `File` is a source name or a list of source names. (toplevel_doc:sourcenames/1)

./2: PREDICATE

Usage: `[File|Files]`

- *Description:* Provided for backward compatibility, obsoleted by `ensure_loaded/1`.
- *The following properties should hold at call time:*

  `File` is a source name. (streams_basic:sourcename/1)

  `Files` is a list of `sourcename`s. (basic_props:list/2)

make_po/1: PREDICATE

Usage: `make_po(Files)`

- *Description:* Make object (`.po`) files from `Files`. Equivalent to executing `"ciaoc -c"` on the files.
- *The following properties should hold at call time:*

  `Files` is a source name or a list of source names. (toplevel_doc:sourcenames/1)

unload/1: PREDICATE

Usage: `unload(File)`

- *Description:* Unloads dynamically loaded file `File`.
- *The following properties should hold at call time:*

  `File` is a source name. (streams_basic:sourcename/1)

set_debug_mode/1: PREDICATE

Usage: `set_debug_mode(File)`

- *Description:* Set the loading mode of `File` to *consult*. See Chapter 6 [The interactive debugger], page 49.
- *The following properties should hold at call time:*

  `File` is a source name. (streams_basic:sourcename/1)

set_nodebug_mode/1: PREDICATE

Usage: `set_nodebug_mode(File)`

- *Description:* Set the loading mode of `File` to *compile*. See Chapter 6 [The interactive debugger], page 49.
- *The following properties should hold at call time:*

  `File` is a source name. (streams_basic:sourcename/1)

make_actmod/2: PREDICATE

Usage: `make_actmod(ModuleFile,PublishMod)`

- *Description:* Make an active module executable from the module residing in `ModuleFile`, using address publish module of name `PublishMod` (which needs to be in the library paths).
- *The following properties should hold at call time:*

  `ModuleFile` is a source name. (streams_basic:sourcename/1)

  `PublishMod` is an atom. (basic_props:atm/1)

force_lazy/1: PREDICATE

Usage: `force_lazy(Module)`

- *Description:* Force module of name `Module` to be loaded lazily in the subsequent created executables.
- *The following properties should hold at call time:*

  `Module` is an atom. (basic_props:atm/1)

undo_force_lazy/1: PREDICATE

Usage: `undo_force_lazy(Module)`

- *Description:* Disable a previous `force_lazy/1` on module `Module` (or, if it is uninstantiated, all previous `force_lazy/1`).
- *Calls should, and exit will be compatible with:*

  `Module` is an atom. (basic_props:atm/1)

dynamic_search_path/1: PREDICATE

Usage: `dynamic_search_path(Name)`

- *Description:* Asserting a fact to this data predicate, files using path alias `Name` will be treated as dynamic in the subsequent created executables.
- *The following properties should hold at call time:*

  `Name` is an atom. (basic_props:atm/1)

multifile/1: PREDICATE

Usage: `multifile Pred`

– *Description:* Dynamically declare predicate `Pred` as multifile. This is useful at the top-level shell to be able to call multifile predicates of loaded files.

– *The following properties should hold at call time:*

`Pred` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

5.6 Documentation on internals (`toplevel_doc`)

sourcenames/1: PROPERTY

Is defined as follows:

```
sourcenames(File) :-
        sourcename(File).
sourcenames(Files) :-
        list(Files,sourcename).
```

See `sourcename/1` in Chapter 21 [Basic file/stream handling], page 159

Usage: `sourcenames(Files)`

– *Description:* `Files` is a source name or a list of source names.

6 The interactive debugger

Author(s): Daniel Cabeza, Manuel C. Rodriguez, Edison Mera, A. Ciepielewski (first version), Mats Carlsson (first version), T. Chikayama (first version), K. Shen (first version).

The Ciao program development environment includes a number of advanced debugging tools, such as a source-level debugger, the `ciaopp` preprocessor, and some execution visualizers. Herein we discuss the interactive debugger available in the standard top-level, which allows tracing the control flow of programs, in a similar way to other popular Prolog systems. This is a classical Byrd *box-type debugger* [Byr80,BBP81], with some enhancements, most notably being able to track the execution on the source program.

We also discuss the embedded debugger, which is a version of the debugger which can be embedded into executables. It allows triggering an interactive debugging session at any time while running an executable, without any need for the top-level shell.

Byrd's Procedure Box model of debugging execution provides a simple way of visualising control flow, including backtracking. Control flow is in principle viewed at the predicate level, rather than at the level of individual clauses. The Ciao debugger has the ability to mark selected modules and/or files for debugging (traditional and source debugging), rather than having to exhaustively trace the program. It also allows to selectively set spy-points and breakpoints. Spy-points allow the programmer to nominate interesting predicates at which program execution is to pause so that the programmer can interact with the debugger. Breakpoints are similar to spy-points, but allow pausing at a specific line in the code, corresponding to a particular literal. There is a wide choice of control and information options available during debugging interaction.

Note: While the debugger described herein can be used in a standalone way (i.e., from an operating system shell or terminal window) in the same way as other Prolog debuggers, the most convenient way of debugging Ciao programs is by using the programming environment (see Chapter 10 [Using Ciao inside GNU emacs], page 67). This environment has many debugging-related facilities, including displaying the source code for the module(s) corresponding to the procedure being executed, and higlighting dynamically the code segments corresponding to the different execution steps.

6.1 Marking modules and files for debugging in the top-level debugger

The Ciao debugger is module-based. This allows skipping during the debugging process all files (including system library files) except those in which a bug is suspected. This saves having to explictily and repetitively skip predicates in unrelated files during the debugging process. Also, there is an efficieny advantage: in order to be able to run the debugger on a module, it must be loaded in *debug (interpreted) mode*, which will execute slower than normal (compiled) modules. Thus, it is interesting to compile in debug mode only those modules that need to be traced. Instead of doing this (loading of modules in one mode or another) by hand each time, in Ciao (re)loading of modules in the appropriate mode is handled automatically by the Ciao compiler. However, this requires the user to *mark explicitly* the modules in which debugging is to be performed. The simplest way of achieving this is by executing in the Ciao shell prompt, for each suspicious module `Module` in the program, the command:

```
?- debug_module(Module).
```

or, alternatively:

```
?- debug_module_source(Module).
```

which in addition instructs the debugger to keep track of the line numbers in the source file and to report them during debugging. This is most useful when running the top-level inside the `emacs` editor since in that case the Ciao emacs mode allows performing full source-level debugging in each module marked as above, i.e., the source lines being executed will be highlighted dynamically during debugging in a window showing the source code of the module.

Note that, since all files with no module declaration belong to the pseudo-module `user`, the command to be issued for debugging a user file, say `foo.pl`, is `debug_module(user)` or `debug_module_source(user)`, and not `debug_module(foo)`.

The two ways of performing source-level debugging are fully compatible between them, i.e., Ciao allows having some modules loaded with `debug_module/1` and others with `debug_module_source/1`. To change from one interpreted mode to the other mode it suffices to select the module with the new interpreted mode (debugger mode), using the appropiate command, and reload the module.

The commands above perform in fact two related actions: first, they let the compiler know that if a file containing a module with this name is loaded, it should be loaded in interpreted mode (source or traditional). In addition, they instruct the debugger to actually prepare for debugging the code belonging to that module. After that, the modules which are to be debugged have to be (re)loaded so that they are compiled or loaded for interpretation in the appropriate way. The nice thing is that, due to the modular behaviour of the compiler/top-level, if the modules are part of a bigger application, it suffices to load the main module of the application, since this will automatically force the dependent modules which have changed to be loaded in the appropriate way, including those whose *loading mode* has changed (i.e., changing the loading mode has the effect of forcing the required re-loading of the module at the appropriate time).

Later in the debugging process, as the bug location is isolated, typically one will want to restrict more and more the modules where debugging takes place. To this end, and without the need for reloading, one can tell the debugger to not consider a module for debugging issuing a `nodebug_module/1` command, which counteracts a `debug_module/1` or `debug_module_source/1` command with the same module name, and reloading it (or the main file).

There are also two top-level commands `set_debug_mode/1` and `set_nodebug_mode/1`, which accept as argument a file spec (i.e., `library(foo)` or `foo`, even if it is a user file) to be able to load a file in interpreted mode without changing the set of modules that the debugger will try to spy.

6.2 The debugging process

Once modules or user files are marked for debugging and reloaded, the traditional debugging shell commands can be used (the documentation of the `debugger` library following this chapter contains all the commands and their description), with the same meaning as in other classical Prolog systems. The differences in their behavior are:

- Debugging takes place only in the modules in which it was activated,
- `nospy/1` and `spy/1` accept sequences of predicate specs, and they will search for those predicates only in the modules marked for debugging (traditional or source-level debugging).
- `breakpt/6` and `nobreakpt/6` allow setting breakpoints at selected clause literals and will search for those literals only in the modules marked for source-level debugging (modules marked with `debug_module_source/1`).

In particular, the system is initially in nodebug mode, in which no tracing is performed. The system can be put in debug mode by a call to `debug/0` in which execution of queries will proceed until the first *spy-point* or *breakpoint*. Alternatively, the system can be put in trace mode by a call to `trace/0` in which all predicates will be trace.

6.3 Marking modules and files for debugging with the embedded debugger

The embedded debugger, as the interpreted debugger, has three different modes of operation: debug, trace or nodebug. These debugger modes can be set by adding *one* of the following package declarations to the module:

```
:- use_package(debug).
:- use_package(trace).
:- use_package(nodebug).
```

and recompiling the application. These declarations *must* appear the last ones of all `use_package` declarations used. Also it is possible, as usual, to add the debugging package(s) in the module declaration using the third argument of the `module/3` declaration (and they should also be the last ones in the list), i.e., using one of:

```
:- module(..., ..., [..., debug]).
:- module(..., ..., [..., trace]).
:- module(..., ..., [..., nodebug]).
```

The nodebug mode allows turning off any debugging (and also the corresponding overhead) but keeping the spy-points and breakpoints in the code. The trace mode will start the debugger for any predicate in the file.

The embedded debugger has limitations over the interpreted debugger. The most important is that the "retry" option is not available. But it is possible to add, and remove, spy-points and breakpoins using the predicates `spy/1`, `nospy/1`, `breakpt/6` and `nobreakpt/6`, etc. These can be used in a clause declaration or as declarations. Also it is possible to add in the code predicates for issuing the debugger (i.e., use debug mode, and in a clause add the predicate `trace/0`). Finally, if a spy declaration is placed on the entry point of an executable (`:- spy(main/1)`) the debugger will not start the first time `main/1` predicate is called, i.e., at the beginning of program execution (however, it will if there are any subsequent calls to `main/1`). Starting the embedded debugger at the beginning of the execution of a program can be done easily however by simply adding the in trace mode.

Note that there is a particularly interesting way of using the embedded debugger: if an *application* is run in a shell buffer which has been set with Ciao inferior mode (⟨M-x⟩ `ciao-inferior-mode`) and this application starts emitting output from the embedded debugger (i.e., which contains the embedded debugger and is debugging its code) then the Ciao emacs mode will be able to follow these messages, for example tracking execution in the source level code. This also works if the application is written in a combination of languages, provided the parts written in Ciao are compiled with the embedded debugger package and is thus a covenient way of debugging multi-language applications. The only thing needed is to make sure that the output messages appear in a shell buffer that is in Ciao inferior mode.

See the following as a general example of use of the embedded debugger:

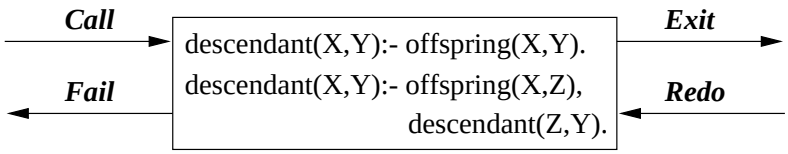
```
:- module( foo,[main/1],[assertions, debug]).

:- entry main/1.

main(X) :-
         display(X),
         spy(foo),
         foo(X),
         notrace,
         nl.

foo([]).
foo([X|T]) :-
        trace,
        bar(X),
        foo(T).

bar(X) :-
```

```
display(X).
```

6.4 The procedure box control flow model

During debugging the interpreter prints out a sequence of goals in various states of instantiation in order to show the state that the program has reached in its execution. However, in order to understand what is occurring it is necessary to understand when and why the interpreter prints out goals. As in other programming languages, key points of interest are procedure entry and return, but in Prolog there is the additional complexity of backtracking. One of the major confusions that novice Prolog programmers have to face is the question of what actually happens when a goal fails and the system suddenly starts backtracking. The Procedure Box model of Prolog execution views program control flow in terms of movement about the program text. This model provides a basis for the debugging mechanism in the interpreter, and enables the user to view the behaviour of the program in a consistent way. It also provides the basis for the visualization performed on the source level program when source level program when source-level debugging is activated within `emacs`.

Let us look at an example Prolog procedure:



The first clause states that `Y` is a descendant of `X` if `Y` is an offspring of `X`, and the second clause states that `Y` is a descendant of `X` if `Z` is an offspring of `X` and `Y` is a descendant of `Z`. In the diagram a box has been drawn around the whole procedure and labelled arrows indicate the control flow in and out of this box. There are four such arrows which we shall look at in turn.

- **Call**

  This arrow represents initial invocation of the procedure. When a goal of the form `descendant(X,Y)` is required to be satisfied, control passes through the Call port of the descendant box with the intention of matching a component clause and then satisfying any subgoals in the body of that clause. Note that this is independent of whether such a match is possible; i.e. first the box is called, and then the attempt to match takes place. Textually we can imagine moving to the code for descendant when meeting a call to descendant in some other part of the code.

- **Exit**

  This arrow represents a successful return from the procedure. This occurs when the initial goal has been unified with one of the component clauses and any subgoals have been satisfied. Control now passes out of the Exit port of the descendant box. Textually we stop following the code for descendant and go back to the place we came from.

- **Redo**

  This arrow indicates that a subsequent goal has failed and that the system is backtracking in an attempt to find alternatives to previous solutions. Control passes through the Redo port of the descendant box. An attempt will now be made to resatisfy one of the component subgoals in the body of the clause that last succeeded; or, if that fails, to completely rematch the original goal with an alternative clause and then try to satisfy any subgoals in the body of this new clause. Textually we follow the code backwards up the way we came looking for new ways of succeeding, possibly dropping down on to another clause and following that if necessary.

- **Fail**

  This arrow represents a failure of the initial goal, which might occur if no clause is matched, or if subgoals are never satisfied, or if any solution produced is always rejected by later

processing. Control now passes out of the Fail port of the descendant box and the system continues to backtrack. Textually we move back to the code which called this procedure and keep moving backwards up the code looking for choice points.

In terms of this model, the information we get about the procedure box is only the control flow through these four ports. This means that at this level we are not concerned with which clause matches, and how any subgoals are satisfied, but rather we only wish to know the initial goal and the final outcome. However, it can be seen that whenever we are trying to satisfy subgoals, what we are actually doing is passing through the ports of *their* respective boxes. If we were following this (e.g., activating source-level debugging), then we would have complete information about the control flow inside the procedure box.

Note that the box we have drawn around the procedure should really be seen as an invocation box. That is, there will be a different box for each different invocation of the procedure. Obviously, with something like a recursive procedure, there will be many different Calls and Exits in the control flow, but these will be for different invocations. Since this might get confusing each invocation box is given a unique integer identifier in the messages, as described below.

Note that not all procedure calls are traced; there are a few basic predicates which have been made invisible since it is more convenient not to trace them. These include debugging directives, basic control structures, and some builtins. This means that messages will never be printed for these predicates during debugging.

6.5 Format of debugging messages

This section explains the two formats of the message output by the debugger at a port. All trace messages are output to the terminal regardless of where the current output stream is directed (which allows tracing programs while they are performing file I/O). The basic format, which will be shown in traditional debug and in source-level debugging within Ciao `emacs` mode, is as follows:

```
   S  13  7  Call: T user:descendant(dani,_123) ?
```

`S` is a spy-point or breakpoint indicator. It is printed as '`+`', indicating that there is a spy-point on `descendant/2` in module `user`, as `'B'` denoting a breakpoint, or as ' ', denoting no spy-point or breakpoint. If there is a spy-point and a breakpoint in the same predicate the spy-point indicator takes preference over breakpoint indicator.

`T` is a subterm trace. This is used in conjunction with the `^` command (set subterm), described below. If a subterm has been selected, `T` is printed as the sequence of commands used to select the subterm. Normally, however, `T` is printed as ' ', indicating that no subterm has been selected.

The first number is the unique invocation identifier. It is always nondecreasing (provided that the debugger is switched on) regardless of whether or not the invocations are being actually seen. This number can be used to cross correlate the trace messages for the various ports, since it is unique for every invocation. It will also give an indication of the number of procedure calls made since the start of the execution. The invocation counter starts again for every fresh execution of a command, and it is also reset when retries (see later) are performed.

The number following this is the *current depth*; i.e., the number of direct *ancestors* this goal has. The next word specifies the particular port (`Call`, `Exit`, `Redo` or `Fail`). The goal is then printed so that its current instantiation state can be inspected. The final `?` is the prompt indicating that the debugger is waiting for user interaction. One of the option codes allowed (see below) can be input at this point.

The second format, quite similar to the format explained above, is shown when using source-level debugging outside the Ciao `emacs` mode, and it is as follows:

```
      In /home/mcarlos/ciao/foo.pl (5-9) descendant-1
   S  13  7  Call: T user:descendant(dani,_123) ?
```

This format is identical to the format above except for the first line, which contains the information for location of the point in the source program text where execution is currently at. The first line contains the name of the source file, the start and end lines where the literal can be found, the substring to search for between those lines and the number of substrings to locate. This information for locating the point on the source file is not shown when executing the source-level debugger from the Ciao `emacs` mode.

Ports can be "unleashed" by calling the `leash/1` predicate omiting that port in the argument. This means that the debugger will stop but user interaction is not possible for an unleashed port. Obviously, the `?` prompt will not be shown in such messages, since the user has specified that no interaction is desired at this point.

6.6 Options available during debugging

This section describes the particular options that are available when the debugger prompts after printing out a debugging message. All the options are one letter mnemonics, some of which can be optionally followed by a decimal integer. They are read from the terminal with any blanks being completely ignored up to the next terminator (carriage-return, line-feed, or escape). Some options only actually require the terminator; e.g., the creep option, only requires ⟨RET⟩.

The only option which really needs to be remembered is 'h' (followed by ⟨RET⟩). This provides help in the form of the following list of available options.

```
<cr>   creep            c      creep
 l     leap             s      skip
 r     retry            r <i>  retry i
 f     fail             f <i>  fail i
 d     display          p      print
 w     write            v <I>  variable(s)
 g     ancestors        g <n>  ancestors n
 n     nodebug          =      debugging
 +     spy this         -      nospy this
 a     abort
 @     command          u      unify
 <     reset printdepth < <n>  set printdepth
 ^     reset subterm    ^ <n>  set subterm
 ?     help             h      help
```

- `c` (*creep*)

  causes the debugger to single-step to the very next port and print a message. Then if the port is leashed the user is prompted for further interaction. Otherwise it continues creeping. If leashing is off, creep is the same as leap (see below) except that a complete trace is printed on the terminal.

- `l` (*leap*)

  causes the interpreter to resume running the program, only stopping when a spy-point or breakpoint is reached (or when the program terminates). Leaping can thus be used to follow the execution at a higher level than exhaustive tracing. All that is needed to do is to set spy-points and breakpoints on an evenly spread set of pertinent predicates or lines, and then follow the control flow through these by leaping from one to the other.

- `s` (*skip*)

  is only valid for Call and Redo ports, if it is issued in Exit or Fail ports it is equivalent to creep. It skips over the entire execution of the predicate. That is, no message will be seen until control comes back to this predicate (at either the Exit port or the Fail port). Skip is particularly useful while creeping since it guarantees that control will be returned after

the (possibly complex) execution within the box. With skip then no message at all will appear until control returns to the Exit port or Fail port corresponding to this Call port or Redo port. This includes calls to predicates with spy-points and breakpoints set: they will be masked out during the skip. There is a way of overriding this: the `t` option after a ⟨^C⟩ interrupt will disable the masking. Normally, however, this masking is just what is required!

- `r` (*retry*)

  can be used at any of the four ports (although at the Call port it has no effect). It transfers control back to the Call port of the box. This allows restarting an invocation when, for example, it has left the programmer with some weird result. The state of execution is exactly the same as in the original call (unless the invocation has performed side effects, which will not be undone). When a retry is performed the invocation counter is reset so that counting will continue from the current invocation number regardless of what happened before the retry. This is in accord with the fact that execution has, in operational terms, returned to the state before anything else was called.

  If an integer is supplied after the retry command, then this is taken as specifying an invocation number and the system tries to get to the Call port, not of the current box, but of the invocation box specified. It does this by continuously failing until it reaches the right place. Unfortunately this process cannot be guaranteed: it may be the case that the invocation the programmer is looking for has been cut out of the search space by cuts in the program. In this case the system fails to the latest surviving Call port before the correct one.

- `f` (*fail*)

  can be used at any of the four ports (although at the Fail port it has no effect). It transfers control to the Fail port of the box, forcing the invocation to fail prematurely. If an integer is supplied after the command, then this is taken as specifying an invocation number and the system tries to get to the Fail port of the invocation box specified. It does this by continuously failing until it reaches the right place. Unfortunately, as before, this process cannot be guaranteed.

- `d` (*display*)

  displays the current goal using `display/1`. See `w` below.

- `p` (*print*)

  re-prints the current goal using `print/1`. Nested structures will be printed to the specified *printdepth* (see below).

- `w` (*write*)

  writes the current goal on the terminal using `write/1`.

- `v` (*variables*)

  writes the list of the modified variables and their values. If a variable name (identifier) `N` is supplied, then the value of variable `N` is shown.

- `g` (*ancestors*)

  provides a list of ancestors to the current goal, i.e., all goals that are hierarchically above the current goal in the calling sequence. It is always possible to jump to any goal in the ancestor list (by using retry, etc.). If an integer `n` is supplied, then only `n` ancestors will be printed. That is to say, the last `n` ancestors will be printed counting back from the current goal. Each entry in the list is preceded by the invocation number followed by the depth number (as would be given in a trace message).

- `n` (*nodebug*)

  switches the debugger off. Note that this is the correct way to switch debugging off at a trace point. The `@` option cannot be used because it always returns to the debugger.

- `=` (*debugging*)

  outputs information concerning the status of the current debugging session.

- `+` *spy*
  sets a spy-point on the current goal.
- `-` (*nospy*)
  removes the spy-point from the current goal.
- `a` (*abort*)
  causes an abort of the current execution. All the execution states built so far are destroyed and the system is put right back at the top-level of the interpreter. (This is the same as the built-in predicate `abort/0`.)
- `@` (*command*)
  allows calling arbitrary goals. The initial message `| ?-` will be output on the terminal, and a command is then read from the terminal and executed as if it was at top-level.
- `u` (*unify*)
  is available at the Call port and gives the option of providing a solution to the goal from the terminal rather than executing the goal. This is convenient, e.g., for providing a "stub" for a predicate that has not yet been written. A prompt `|:` will be output on the terminal, and the solution is then read from the terminal and unified with the goal.
- `<` (*printdepth*)
  sets a limit for the subterm nesting level that is printed in messages. While in the debugger, a printdepth is in effect for limiting the subterm nesting level when printing the current goal. When displaying or writing the current goal, all nesting levels are shown. The limit is initially 10. This command, without arguments, resets the limit to 10. With an argument of `n` the limit is set to `n`.
- `^` (*subterm*)
  sets the subterm to be printed in messages. While at a particular port, a current subterm of the current goal is maintained. It is the current subterm which is displayed, printed, or written when prompting for a debugger command. Used in combination with the printdepth, this provides a means for navigating in the current goal for focusing on the part which is of interest. The current subterm is set to the current goal when arriving at a new port. This command, without arguments, resets the current subterm to the current goal. With an argument of `n` (greater than 0 and less or equal to the number of subterms of the current subterm), the current subterm is replaced by its `n`'th subterm. With an argument of `0`, the current subterm is replaced by its parent term.
- `?` or `h` (*help*)
  displays the table of options given above.

6.7 Calling predicates that are not exported by a module

The Ciao module system does not allow calling predicates which are not exported during debugging. However, as an aid during debugging, this is allowed (only from the top-level and for modules which are in debug mode or source-level debug mode) using the `call_in_module/2` predicate.

Note that this does not affect analysis or optimization issues, since it only works on modules which are loaded in debug mode or source-level debug mode, i.e. unoptimized.

6.8 Acknowledgements (debugger)

Originally written by Andrzej Ciepielewski. Minor modifications by Mats Carlsson. Later modifications (17 Dec 87) by Takashi Chikayama (making tracer to use `print/1` rather than `write/1`, temporarily switching debugging flag off while writing trace message and within

"break" level). Additional modifications by Kish Shen (May 88): subterm navigation, handle unbound args in `spy/1` and `nospy/1`, trapping arithmetics errors in debug mode. Adapted then to &-Prolog and Ciao by Daniel Cabeza and included in the Ciao version control system. Extended for source-level debugging by Manuel C. Rodríguez. Option that shows variable names and values (`v <N>`) implemented by Edison Mera (2009). (See changelog if included in the document or in the version maintenance system for more detailed documentation on changes.)

7 Predicates controlling the interactive debugger

Author(s): A. Ciepielewski, Mats Carlsson, T. Chikayama, K. Shen, Daniel Cabeza, Manuel C. Rodriguez, Edison Mera.

This library implements predicates which are normally used in the interactive top-level shell to debug programs. A subset of them are available in the embeddable debugger.

7.1 Usage and interface (`debugger`)

- **Library usage:**
  `:- use_module(library(debugger)).`
- **Exports:**
  - *Predicates:*
    `call_in_module/2`.
  - *Multifiles:*
    `define_flag/3`.
- **Other modules used:**
  - *System library modules:*
    `debugger/debugger_lib`, `format`, `ttyout`.

7.2 Documentation on exports (`debugger`)

call_in_module/2: PREDICATE

Usage: `call_in_module(Module,Predicate)`

- *Description:* Calls predicate `Predicate` belonging to module `Module`, even if that module does not export the predicate. This only works for modules which are in debug (interpreted) mode (i.e., they are not optimized).
- *The following properties should hold at call time:*
  `Module` is an atom. (basic_props:atm/1)
  `Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

breakpt/6: (UNDOC_REEXPORT)

Imported from `debugger_lib` (see the corresponding documentation for details).

debug/0: (UNDOC_REEXPORT)

Imported from `debugger_lib` (see the corresponding documentation for details).

debug_module/1: (UNDOC_REEXPORT)

Imported from `debugger_lib` (see the corresponding documentation for details).

debug_module_source/1: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

debugging/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

debugrtc/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

get_debugger_state/1: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

get_debugger_state/1: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

leash/1: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

list_breakpt/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

maxdepth/1: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

nobreakall/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

nobreakpt/6: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

nodebug/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

nodebug_module/1: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

nodebugrtc/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

nospy/1: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

nospyall/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

notrace/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

spy/1: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

trace/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

tracertc/0: (UNDOC_REEXPORT)
Imported from `debugger_lib` (see the corresponding documentation for details).

7.3 Documentation on multifiles (`debugger`)

define_flag/3: PREDICATE
The predicate is *multifile*.
Usage: `define_flag(Flag,FlagValues,Default)`
- *The following properties hold upon exit:*
  `Flag` is an atom. (basic_props:atm/1)
  Define the valid flag values (basic_props:flag_values/1)

7.4 Known bugs and planned improvements (`debugger`)

- Add an option to the emacs menu to automatically select all modules in a project.
- Consider the possibility to show debugging messages directly in the source code emacs buffer.

8 The script interpreter

Author(s): Daniel Cabeza, Manuel Hermenegildo.

`ciao-shell` is the Ciao script interpreter. It can be used to write *Prolog shell scripts* (see [Her96,CHV96b]), that is, executable files containing source code, which are compiled on demand.

Writing Prolog scripts can sometimes be advantageous with respect to creating binary executables for small- to medium-sized programs that are modified often and perform relatively simple tasks. The advantage is that no explicit compilation is necessary, and thus changes and updates to the program imply only editing the source file. The disadvantage is that startup of the script (the first time after it is modified) is slower than for an application that has been compiled previously.

An area of application is, for example, writing *CGI executables*: the slow speed of the network connection in comparison with that of executing a program makes program execution speed less important and has made scripting languages very popular for writing these applications. Logic languages are, a priori, excellent candidates to be used as scripting languages. For example, the built-in grammars and databases can sometimes greatly simplify many typical script-based applications.

8.1 How it works

Essentially, `ciao-shell` is a smaller version of the Ciao top-level, which starts by loading the file given to it as the first argument and then starts execution at `main/1` (the argument is instantiated to a list containing the command line options, in the usual way). Note that the Prolog script cannot have a `module` declaration for this to work. While loading the file, `ciao-shell` changes the prolog flag `quiet` so that no informational or warning messages are printed (error messages will be reported to `user_error`, however). The operation of `ciao-shell` in Unix-like systems is based in a special compiler feature: when the first character of a file is '`#`', the compiler skips the first lines until an empty line is found. In Windows, its use is as easy as naming the file with a `.pls` extension, which will launch `ciao-shell` appropriately.

For example, in a Linux/Unix system, assume a file called `hello` contains the following program:

```
#!/bin/sh
exec ciao-shell $0 "$@" # -*- mode: ciao; -*-

main(_) :-
     write('Hello world'), nl.
```

Then, the file `hello` can be *run* by simply making it executable and invoking it from the command line:

```
/herme@clip:/tmp
[86]> chmod +x hello

/herme@clip:/tmp
[87]> hello
Hello world
```

The line:

```
#!/bin/sh
```

invokes the `/bin/sh` shell which will interpret the following line:

```
exec ciao-shell $0 "$@" # -*- mode: ciao; -*-
```

and invoke `ciao-shell`, instructing it to read this same file (`$0`), passing it the rest of the arguments to `hello` as arguments to the prolog program. The second part of the line `# -*- mode: ciao; -*-` is simply a comment which is seen by `emacs` and instructs it to edit this file in Ciao mode (this is needed because these script files typically do not have a `.pl` ending). When `ciao-shell` starts, if it is the first time, it compiles the program (skipping the first lines, as explained above), or else at successive runs loads the `.po` object file, and then calls `main/1`.

Note that the process of creating Prolog scripts is made very simple by the Ciao emacs mode, which automatically inserts the header and makes the file executable (See Chapter 10 [Using Ciao inside GNU emacs], page 67).

8.2 Command line arguments in scripts

The following example illustrates the use of command-line arguments in scripts. Assume that a file called `say` contains the following lines:

```
#!/bin/sh
exec ciao-shell $0 "$@" # -*- mode: ciao; -*-

main(Argv) :-
     write_list(Argv), nl.

write_list([]).
write_list([Arg|Args]) :-
     write(Arg),
     write(' '),
     write_list(Args).
```

An example of use is:

```
/herme@clip:/tmp
[91]> say hello dolly
hello dolly
```

9 Customizing library paths and path aliases

Author(s): Daniel Cabeza.

This library provides means for customizing, from environment variables, the libraries and path aliases known by an executable. Many applications of Ciao, including `ciaoc`, `ciaosh`, and `ciao-shell` make use of this library. Note that if an executable is created dynamic, it will try to load its components at startup, before the procedures of this module can be invoked, so in this case all the components should be in standard locations.

9.1 Usage and interface (`libpaths`)

- **Library usage:**
  `:- use_module(library(libpaths)).`
- **Exports:**
  - *Predicates:*
    `get_alias_path/0`.
  - *Multifiles:*
    `file_search_path/2`, `library_directory/1`.
- **Other modules used:**
  - *System library modules:*
    `system`, `lists`.

9.2 Documentation on exports (`libpaths`)

get_alias_path/0: PREDICATE

`get_alias_path`

Consult the environment variable 'CIAOALIASPATH' and add facts to predicates `library_directory/1` and `file_search_path/2` to define new library paths and path aliases. The format of 'CIAOALIASPATH' is a sequence of paths or alias assignments separated by colons, an alias assignment is the name of the alias, an '=' and the path represented by that alias (no blanks allowed). For example, given

```
CIAOALIASPATH=/home/bardo/ciao:contrib=/usr/local/lib/ciao
```

the predicate will define /home/bardo/ciao as a library path and /usr/local/lib/ciao as the path represented by 'contrib'.

9.3 Documentation on multifiles (`libpaths`)

file_search_path/2: PREDICATE

See Chapter 21 [Basic file/stream handling], page 159.

The predicate is *multifile*.

The predicate is of type *dynamic*.

`file_search_path(X,Y)`

– *The following properties hold upon exit:*
`X` is ground. (basic_props:gnd/1)
`Y` is ground. (basic_props:gnd/1)

library_directory/1: PREDICATE

See Chapter 21 [Basic file/stream handling], page 159.

The predicate is *multifile*.

The predicate is of type *dynamic*.

```
library_directory(X)
```

– *The following properties hold upon exit:*
`X` is ground. (basic_props:gnd/1)

10 Using Ciao inside GNU emacs

Author(s): Manuel Hermenegildo, Manuel C. Rodriguez, Daniel Cabeza.

The Ciao emacs interface (or *mode* in `emacs` terms) provides a rich, integrated user interface to the Ciao *program development environment* components, including the `ciaosh` interactive top level and the `ciaopp` preprocessor. While most features of the Ciao development environment are available from the command line of the preprocessor and the top-level shell, using Ciao inside `emacs` is highly recommended. The facilities that this mode provides include:

- *Syntax-based highlighting* (coloring), *auto-indentation*, *auto-fill*, etc. of code. This includes the assertions used by the preprocessor and the documentation strings used by the Ciao auto-documenter, `lpdoc`.
- Providing automatic access to on-line help for all predicates by accessing the Ciao system manuals in `info` format.
- Starting and communicating with `ciaopp`, the *Ciao preprocessor*, running in its own sub-shell. This allows easily performing certain kinds of *static checks* (useful for finding errors in programs before running them), program analysis tasks, and *program transformations* on source programs.
- Starting and communicating with the *Ciao top-level*, running in its own sub-shell. This facilitates loading programs, checking the *syntax* of programs (and of *assertions* within programs), marking and unmarking modules for interactive debugging, *tracing the source code* during debugging, making stand-alone executables, compiling modules to dynamically linkable Ciao objects, compiling modules to active objects, etc.
- Syntax highlighting and coloring of the error and warning messages produced by the top level, preprocessor, or any other tool using the same message format (such as the `lpdoc` auto-documenter), and *locating automatically the points in the source files where such errors occur*.
- Performing automatic *version control* and keeping a *changelog* of individual files or whole applications. This is done by automatically including changelog entries in source files, which can then be processed by the `lpdoc` auto-documenter.

This chapter explains how to use the Ciao `emacs` interface and how to set up your `emacs` environment for correct operation. The Ciao `emacs` interface can also be used to work with traditional Prolog or CLP systems.

10.1 Conventions for writing Ciao programs under Emacs

This is particularly important for the source-level debugger and the syntax-based coloring capabilities. This is due to the fact that it would be unrealistic to write a complete Ciao parser in Emacs lisp. These conventions are the following, in order of importance:

- Clauses should begin on the first column (this is used to recognize the beginning of a clause).
- C style comments should not be used in a clause, but can be used outside any clause.

The following suggestion is not strictly necessary but can improve operation:

- Body literals should be indented. There should be not more than one literal per line. This allows more precision in the location of program points during source-level debugging, i.e., when marking breakpoints and during line tracing.

Comments which start with `%`s are indented to the right if indentation is asked for.

For syntax-based highlighting to be performed font-lock must be available and not disabled (the Ciao mode enables it but it may be disabled elsewhere in, e.g., the `.emacs` file).

10.2 Checking the installation

Typically, a complete pre-installation of the Ciao `emacs` interface is completed during Ciao installation. To check that installation was done and sucessful, open a file with a `.pl` ending. You should see that `emacs` enters Ciao mode: the mode is identified in the status bar below the buffer and, if the emacs menu bar is enabled, you should see the Ciao menus. You should be able from the menu-bar, for example, to go to the Ciao manuals in the info or load the `.pl` file that you just opened into a Ciao top level.

If things don't work properly, see the section Section 10.21 [Installation of the Ciao emacs interface], page 85 later in this chapter.

10.3 Functionality and associated key sequences (bindings)

The following sections summarize the capabilities of the Ciao emacs interface and the (default) *key sequences* used to access those capabilities. Most of these functions are accessible also from the menu bar.

10.4 Syntax coloring and syntax-based editing

Syntax-based highlighting (coloring) of code is provided automatically when opening Ciao files. This includes also the assertions used by the preprocessor and the documentation strings used by the Ciao auto-documenter, `lpdoc`. The mode should be set to Ciao and the Ciao mode menus should appear on the menu bar. The colors and fonts used can be changed through the *customize* options in the help menu (see Section 10.20 [Customization], page 80).

During editing this coloring may be refreshed by calling the appropriate function (see below).

Limited syntax-based auto-indentation and auto-fill of code and comments is also provided. Syntax highlighting and coloring is also available for the error and warning messages produced by the top level, preprocessor, and auto-documenter, and, in general, for the output produced by these tools.

Commands:

⟨C-c⟩ ⟨h⟩ Undate (recompute) syntax-based highlighting (coloring).

⟨TAB⟩ Indent current line as Ciao code. With argument, indent any additional lines of the same clause rigidly along with this one.

10.5 Getting on-line help

The following commands are useful for getting on-line help. This is done by accessing the `info` version of the Ciao manuals or the `emacs` built-in help strings. Note also that the `info` standard `search` command (generally bound to ⟨s⟩) can be used inside `info` buffers to search for a given string.

⟨C-c⟩ ⟨TAB⟩ Find help for the symbol (e.g., predicate, directive, declaration, type, etc.) that is currently under the cursor. Opens a (hopefully) relevant part of the Ciao manuals in `info` mode. Requires that the Ciao manuals in `info` format be installed and accessible to `emacs` (i.e., they should appear somewhere in the info directory when typing `M-x info`). It also requires `word-help.el`, which is provided with Ciao. Refer to the installation instructions if this is not the case.

⟨C-c⟩ ⟨/⟩ Find a completion for the symbol (e.g., predicate, directive, declaration, type, etc.) that is currently under the cursor. Uses for completion the contents of the indices of the Ciao manuals. Same requirements as for finding help for the symbol.

⟨C-c⟩ ⟨RET⟩ Go to the part of the info directory containing the Ciao manuals.

⟨C-h⟩ ⟨m⟩ Show a short description of the Ciao emacs mode, including all key bindings.

10.6 Loading and compiling programs

These commands allow *loading programs*, *creating executables*, etc. by issuing the appropriate commands to a Ciao top level shell, running in its own buffer as a subprocess. See Chapter 5 [The interactive top-level shell], page 41 for details. The following commands implement the communication with the Ciao top level:

⟨C-c⟩ ⟨t⟩ Ensure that an inferior Ciao top-level process is running.

This opens a top-level window (if one did not exist already) where queries can be input directly. Programs can be loaded into this top level by typing the corresponding commands in this window (such as use_module, etc.), or, more typically, by opening the file to be loaded in an emacs window (where it can be edited) and issuing a load command (such as C-c l or C-c L) directly from there (see the loading commands of this mode and their bindings).

Note that many useful commands (e.g., to repeat and edit previous commands, interrupt jobs, locate errors, automatic completions, etc.) are available in this top-level window (see Section 10.7 [Commands available in toplevel and preprocessor buffers], page 70).

Often, it is not necessary to use this function since execution of any of the other functions related to the top level (e.g., loading buffers into the top level) ensures that a top level is started (starting one if required).

⟨C-c⟩ ⟨l⟩ Load the current buffer (and any auxiliary files it may use) into the top level.

The type of compilation performed (*compiling* or *interpreting*) is selected automatically depending on whether the buffer has been marked for debugging or not – see below. In case you try to load a file while in the middle of the debugging process the debugger is first aborted and then the buffer is loaded. Also, if there is a defined query, the user is asked whether it should be called.

⟨C-c⟩ ⟨f⟩ Load CiaoPP and then the current buffer (and any auxiliary files it may use) into the top level. Use CiaoPP auto_check_assrt predicate to check current buffer assertions and then load the buffer if there was no error.

⟨C-c⟩ ⟨x⟩ Make an executable from the code in the current buffer. The buffer must contain a `main/0` or `main/1` predicate. Note that compiler options can be set to determine whether the libraries and auxiliary files used by the executable will be statically linked, dynamically linked, auto-loaded, etc.

⟨C-c⟩ ⟨o⟩ Make a Ciao object (.po) file from the code in the current buffer. This is useful for example while debugging during development of a very large application which is compiled into an excutable, and only one or a few files are modified. If the application executable is dynamically linked, i.e., the component .po files are loaded dynamically during startup of the application, then this command can be used to recompile only the file or files which have changed, and the correct version will be loaded dynamically the next time the application is started. However, note that this must be done with care since it only works if the inter-module interfaces have not changed. The recommended, much safer way is to generate the executable again, letting the Ciao compiler, which is inherently incremental, determine what needs to be recompiled.

⟨C-c⟩ ⟨a⟩ Make an active module executable from the code in the current buffer. An active module is a remote procedure call server (see the `activemod` library documentation for details).

⟨C-c⟩ ⟨s⟩ Set the current buffer as the principal file in a multiple module programming environment.

⟨C-c⟩ ⟨L⟩ Load the module designated as *main module* (and all related files that it uses) into the top level. If no main module is defined it will load the current buffer.

The type of compilation performed (*compiling* or *interpreting*) is selected automatically depending on whether the buffer has been marked for debugging or not – see below. In case you try to load a file while in the middle of the debugging process the debugger is first aborted and then the buffer is loaded. Also, if there is a defined query, the user is asked whether is should be called.

⟨C-c⟩ ⟨q⟩ Set a default query. This may be useful specially during debugging sessions. However, as mentioned elsewhere, note that commands that repeat previous queries are also available.

This query can be recalled at any time using C-c Q. It is also possible to set things up so that this query will be issued automatically any time a program is (re)loaded. The functionality is available in the major mode (i.e., from a buffer containing a source file) and in the inferior mode (i.e., from the buffer running the top-level shell). When called from the major mode (i.e., from window containing a source file) then the user is prompted in the minibuffer for the query. When called from the inferior mode (i.e., from a top-level window) then the query on the current line, following the Ciao prompt, is taken as the default query.

To clear the default query use M-x ciao-clear-query or simply set it to an empty query: i.e., in a source buffer select C-c q and enter an empty query. In an inferior mode simply select C-c q on a line that contains only the system prompt.

⟨C-c⟩ ⟨Q⟩ Issue predefined query.

10.7 Commands available in toplevel and preprocessor buffers

The interactive top level and the preprocessor both are typically run in an iteractive buffer, in which it is possible to communicate with them in the same way as if they had been started from a standard shell. These interactive buffers run in the so-called *Ciao inferior mode*. This is a particular version of the standard emacs shell package (comint) and thus all the commands typically available when running shells inside emacs also work in these buffers. In addition, many of the commands and key bindings available in buffers containing Ciao source code are also available in these interactive buffers, when applicable. The Ciao-specific commands available include:

⟨C-c⟩ ⟨TAB⟩ Find help for the symbol (e.g., predicate, directive, declaration, type, etc.) that is currently under the cursor. Opens a (hopefully) relevant part of the Ciao manuals in `info` mode. Requires that the Ciao manuals in `info` format be installed and accessible to `emacs` (i.e., they should appear somewhere in the info directory when typing `M-x info`). It also requires `word-help.el`, which is provided with Ciao. Refer to the installation instructions if this is not the case.

⟨C-c⟩ ⟨/⟩ Find a completion for the symbol (e.g., predicate, directive, declaration, type, etc.) that is currently under the cursor. Uses for completion the contents of the indices of the Ciao manuals. Same requirements as for finding help for the symbol.

⟨C-c⟩ ⟨`⟩ Go to the location in the source file containing the next error reported by the last Ciao subprocess (preprocessor or toplevel) which was run.

⟨C-c⟩ ⟨e⟩ Remove error marks from last run (and also debugging marks if present).

⟨C-c⟩ ⟨q⟩ Set a default query. This may be useful specially during debugging sessions. However, as mentioned elsewhere, note that commands that repeat previous queries are also available.

This query can be recalled at any time using C-c Q. It is also possible to set things up so that this query will be issued automatically any time a program is (re)loaded.

The functionality is available in the major mode (i.e., from a buffer containing a source file) and in the inferior mode (i.e., from the buffer running the top-level shell). When called from the major mode (i.e., from window containing a source file) then the user is prompted in the minibuffer for the query. When called from the inferior mode (i.e., from a top-level window) then the query on the current line, following the Ciao prompt, is taken as the default query.

To clear the default query use M-x ciao-clear-query or simply set it to an empty query: i.e., in a source buffer select C-c q and enter an empty query. In an inferior mode simply select C-c q on a line that contains only the system prompt.

⟨C-c⟩ ⟨Q⟩ Issue predefined query.

⟨C-c⟩ ⟨C-v⟩ Show last output file produced by Ciao preprocessor. The preprocessor works by producing a file which is a transformed and/or adorned (with assertions) version of the input file. This command is often used after running the preprocessor in order to visit the output file and see the results from running the preprocessor.

⟨C-c⟩ ⟨v⟩ Report the version of the emacs Ciao mode.

The following are some of the commands from the comint shell package which may be specially useful (type `<f1> m` while in a Ciao interactive buffer for a complete list of commands):

⟨M-p⟩ Cycle backwards through input history, saving input.

⟨M-n⟩ Cycle forwards through input history.

⟨M-r⟩ Search backwards through input history for match for REGEXP. (Previous history elements are earlier commands.) With prefix argument N, search for Nth previous match. If N is negative, find the next or Nth next match.

⟨TAB⟩ Dynamically find completion of the item at point. Note that this completion command refers generally to filenames (rather than, e.g., predicate names, as in the previous functions).

⟨M-?⟩ List all (filename) completions of the item at point.

⟨RET⟩ Return at any point of the a line at the end of a buffer sends that line as input. Return not at end copies the rest of the current line to the end of the buffer and sends it as input.

⟨^D⟩ Delete ARG characters forward or send an EOF to subprocess. Sends an EOF only if point is at the end of the buffer and there is no input.

⟨^C⟩ ⟨^U⟩ Kill all text from last stuff output by interpreter to point.

⟨^C⟩ ⟨^W⟩ Kill characters backward until encountering the beginning of a word. With argument ARG, do this that many times.

⟨^C⟩ ⟨^C⟩ Interrupt the current subjob. This command also kills the pending input between the process mark and point.

⟨^C⟩ ⟨^Z⟩ Stop the current subjob. This command also kills the pending input between the process mark and point.

WARNING: if there is no current subjob, you can end up suspending the top-level process running in the buffer. If you accidentally do this, use M-x comint-continue-subjob to resume the process. (This is not a problem with most shells, since they ignore this signal.)

⟨^C⟩ ⟨^\⟩ Send quit signal to the current subjob. This command also kills the pending input between the process mark and point.

10.8 Locating errors and checking the syntax of assertions

These commands allow locating quickly the point in the source code corresponding to errors flagged by the compiler or preprocessor as well as performing several syntactic checks of assertions:

⟨C-c⟩ ⟨`⟩ Go to the location in the source file containing the next error reported by the last Ciao subprocess (preprocessor or toplevel) which was run.

⟨C-c⟩ ⟨e⟩ Remove error marks from last run (and also debugging marks if present).

⟨C-c⟩ ⟨E⟩ Check the *syntax* of the code and assertions in the current buffer, as well as imports and exports. This uses the standard top level (i.e., does not call the preprocessor and thus does not require the preprocessor to be installed). Note that full (semantic) assertion checking must be done with the preprocessor.

10.9 Commands which help typing in programs

The following commands are intended to help in the process of writing programs:

⟨C-c⟩ ⟨I⟩ ⟨S⟩ Insert a (Unix) header at the top of the current buffer so that the Ciao script interpreter will be called on this file if *run* from the command line. It also makes the file "executable" (e.g., '`chmod +x <file>`' in Unix). See Chapter 8 [The script interpreter], page 63 for details.

⟨C-c⟩ ⟨i⟩ Indent a Ciao or Prolog file.

10.10 Debugging programs

These commands allow marking modules for *debugging* by issuing the appropiate commands to a Ciao top level shell, running in its own buffer as a subprocess. There are two differents types of debugging: traditional debugging (using the byrd-box model and spy-points) and *source-level debugging* (same as traditional debugging plus source tracing and breakpoints). In order to use *breakpoints*, source debugging must be on. The following commands implement comunication with the Ciao top level:

⟨C-c⟩ ⟨d⟩ Debug (or stop debugging) buffer source. This is a shortcut which is particularly useful when using the source debugger on a single module. It corresponds to several lower-level actions. Those lower-level actions depend on how the module was selected for debugging. In case the module was not marked for source-level debugging, it marks the module corresponding to the current buffer for source-level debugging, reloads it to make sure that it is loaded in the correct way for debugging (same as C-c l), and sets the debugger in trace mode (i.e., issues the `trace.` command to the top-level shell). Conversely, if the module was already marked for source-level debugging then it will take the opposite actions, i.e., it unmarks the module for source-level debugging, reloads it, and sets the debugger to non-debug mode.

⟨C-c⟩ ⟨m⟩ Mark, or unmark, the current buffer for debugging (traditional debugging or source debugging). Note that if the buffer has already been loaded while it was unmarked for debugging (and has therefore been loaded in "compile" mode) it has to be loaded again. The minibuffer shows how the module is loaded now and allows selecting another mode for it. There are three posibilities: N for no debug, S for source debug and D for traditional debug.

⟨C-c⟩ ⟨M-m⟩ Visits all Ciao files which are currently open in a buffer allowing selecting for each of them whether to debug them or not and the type of debugging performed. When working on a multiple module program, it is possible to have many modules open

at a time. In this case, you will navigate through all open Ciao files and select the debug mode for each of them (same as doing C-c m for each).

⟨C-c⟩ ⟨S⟩ ⟨b⟩ Set a breakpoint on the current literal (goal). This can be done at any time (while debugging or not). The cursor must be *on the predicate symbol of the literal*. Breakpoints are only useful when using source-level debugging.

⟨C-c⟩ ⟨S⟩ ⟨v⟩ Remove a breakpoint from the current literal (goal). This can be done at any time (while debugging or not). The cursor must be *on the predicate symbol of the literal*.

⟨C-c⟩ ⟨S⟩ ⟨n⟩ Remove all breakpoints. This can be done at any time (while debugging or not).

⟨C-c⟩ ⟨S⟩ ⟨l⟩ Redisplay breakpoints in all Ciao buffers. This ensures that the marks in the source files and the Ciao toplevel are synchronized.

⟨C-c⟩ ⟨S⟩ ⟨r⟩ Remove breakpoint coloring in all Ciao files.

⟨C-c⟩ ⟨S⟩ ⟨t⟩ Set the debugger to the trace state. In this state, the program is executed step by step.

⟨C-c⟩ ⟨S⟩ ⟨d⟩ Set the debugger to the debug state. In this state, the program will only stop in breakpoints and spypoints. Breakpoints are specially supported in `emacs` and using source debug.

⟨C-c⟩ ⟨r⟩ Load the current region (between the cursor and a previous mark) into the top level. Since loading a region of a file is typically done for debugging and/or testing purposes, this command always loads the region in debugging mode (interpreted).

⟨C-c⟩ ⟨p⟩ Load the predicate around the cursor into the top level. Since loading a single predicate is typically done for debugging and/or testing purposes, this command always loads the predicate in debugging mode (interpreted).

⟨C-c⟩ ⟨u⟩ Run the test over the current buffer.

The test should be specified using a test assertion in the module.

10.11 Preprocessing programs

These commands allow *preprocessing programs* with `ciaopp`, the *Ciao preprocessor*.

CiaoPP is the abstract interpretation-based preprocessor of the Ciao multi-paradigm program development environment. CiaoPP can perform a number of program debugging, analysis, and source-to-source transformation tasks on (Ciao) Prolog programs. These tasks include:

- Inference of properties of the predicates and literals of the program, including types, modes and other variable instantiation properties, non-failure, determinacy, bounds on computational cost, bounds on sizes of terms in the program, etc.
- Certain kinds of static debugging and verification, finding errors before running the program. This includes checking how programs call system library predicates and also checking the assertions present in the program or in other modules used by the program. Such assertions represent essentially partial specifications of the program.
- Several kinds of source to source program transformations such as program specialization, slicing, partial evaluation of a program, program parallelization (taking granularity control into account), inclusion of run-time tests for assertions which cannot be checked completely at compile-time, etc.
- The abstract model of the program inferred by the analyzers is used in the system to certify that an untrusted mobile code is safe w.r.t. the given policy (i.e., an abstraction-carrying code approach to mobile code safety).

The information generated by analysis, the assertions in the system specifications are all written in the same assertion language, which is in turn also used by the Ciao system documentation generator, `lpdoc`.

CiaoPP is distributed under the GNU general public license.

See the preprocessor manual for details. The following commands implement the communication with the Ciao preprocessor:

⟨C-c⟩ ⟨A⟩ Call the preprocessor to perform a number of pre-selected analyses on the current buffer (and related modules).

⟨C-c⟩ ⟨T⟩ Call the preprocessor to perform compile-time checking of the assertions (types, modes, determinacy, nonfailure, cost, ...) in the current buffer (and against those in related modules).

⟨C-c⟩ ⟨O⟩ Uses the preprocessor to perform optimizations (partial evaluation, abstract specialization, parallelization, ...) on the current buffer (and related modules).

⟨C-c⟩ ⟨M⟩ Browse and select (using the preprocessor menus) the actions to be performed by the preprocessor when performing analisys used by M-x ciao- C-c A, C-c T, C-c O, and the corresponding toolbar buttons.

⟨C-c⟩ ⟨C-v⟩ Show last output file produced by Ciao preprocessor. The preprocessor works by producing a file which is a transformed and/or adorned (with assertions) version of the input file. This command is often used after running the preprocessor in order to visit the output file and see the results from running the preprocessor.

⟨C-c⟩ ⟨C-r⟩ Ensure that an inferior Ciao preprocessor process is running.

This opens a preprocessor top-level window (if one did not exist already) where preprocessing commands and preprocessing menu options can be input directly. Programs can be preprocessed by typing commands in this window, or, more typically, by opening the file to be preprocessed in an emacs window (where it can be edited) and issuing a command (such as C-c A, C-c T, C-c O, or C-c M) directly from there (see the preprocessing commands of this mode and their bindings).

Note that many useful commands (e.g., to repeat and edit previous commands, interrupt jobs, locate errors, automatic completions, etc.) are available in this top-level window (see Section 10.7 [Commands available in toplevel and preprocessor buffers], page 70).

Often, it is not necessary to use this function since execution of any of the other functions related to the top level (e.g., loading buffers into the top level) ensures that a top level is started (starting one if required).

10.12 Version control

The following commands can be used to carry out a simple but effective form of version control by keeping a log of changes on a file or a group of related files. Interestingly, this log is kept in a format that is understood by `lpdoc`, the Ciao documenter [Her99]. As a result, if these version comments are present, then `lpdoc` will be able to automatically assign up to date version numbers to the manuals that it generates. This way it is always possible to identify to which version of the software a manual corresponds. Also, `lpdoc` can create automatically sections describing the changes made since previous versions, which are extracted from the comments in the changelog entries.

The main effect of these commands is to automatically associate the following information to a set of changes performed in the file and/or in a set of related files:

- a *version number* (such as, e.g., `1.2`, where `1` is the major version number and `2` is the minor version number),

- a patch number (such as, e.g., the `4` in `1.2#4`),
- a time stamp (such as, e.g., `1998/12/14,17:20*28+MET`),
- the author of the change, and
- a comment explaining the change.

The version numbering used can be local to a single file or common to a number of related files. A simple version numbering policy is implemented: when a relevant change is made, the user typically inserts a changelog entry for it, using the appropriate command (or selecting the corresponding option when prompted while saving a file). This will cause the *patch number* for the file (or for the whole system that the file is part of) to be incremented automatically and the corresponding machine-readable comment to be inserted in the file. Major and minor version numbers can also be changed, but this is always invoked by hand (see below).

The changelog entry is written in the form of a `comment/2` declaration. As mentioned before, the advantage of using this kind of changelog entries is that these declarations can be processed by the `lpdoc` automatic documenter (see the `lpdoc` reference manual [Her99] or the `assertions` library documentation for more details on these declarations).

Whether the user is asked or not to introduce such changelog entries, and how the patch and version numbers should be increased is controlled by the presence in the file of a `comment/2` declaration of the type:

```
:- doc(version_maintenance,<type>).
```

(note that this requires including the `assertions` library in the source file). These declarations themselves are also typically introduced automatically when using this mode (see below).

The version maintenance mode can also be set alternatively by inserting a comment such as:

```
%% Local Variables:
%% mode: ciao
%% update-version-comments: "off"
%% End:
```

The lines above instruct emacs to put the buffer visiting the file in emacs Ciao mode and to turn version maintenance off. Setting the version maintenance mode in this way has the disadvantage that `lpdoc`, the auto-documenter, and other related tools will not be aware of the type of version maintenance being performed (the lines above are comments for Ciao). However, this can be useful in fact for setting the *version maintenance mode for packages* and other files meant for inclusion in other files, since that way the settings will not affect the file in which the package is included.

The following commands implement the version control support:

⟨C-c⟩ ⟨C-a⟩ Used to turn on or off version control for the file being visited in the current buffer. The user will be prompted to choose among the following options:

- ⟨y⟩ Turn version control on for this file.
- ⟨n⟩ Turn version control off for this file. A version control comment such as:

  ```
  :- doc(version_maintenance,off).
  ```

  will be added to the buffer and the file saved. No version control will be performed on this file until the line above is removed or modified (i.e., from now on C-x C-s simply saves the buffer).
- ⟨q⟩ Turn off prompting for the introduction of changelog entries for now. `emacs` will not ask again while the buffer is loaded, but it may ask again when saving after the next time you load the buffer (if `ciao-ask-for-version-maintenance-type` is set to `yes`).

If ⟨y⟩ is selected, then the system prompts again regarding how and where the version and patch number information is to be maintained. The following options are available:

`on` All version control information will be contained within this file. When saving a buffer (C-x C-s) emacs will ask if a changelog entry should be added to the file before saving. If a comment is entered by the user, a new patch number is assigned to it and the comment is added to the file. This patch number will be the one that follows the most recent changelog entry already in the file. This is obviously useful when maintaining version numbers individually for each file.

`<directory_name>`
Global version control will be performed coherently on several files. When saving a buffer (C-x C-s) emacs will ask if a changelog entry should be added to the file before saving. If a comment is given, the global patch number (which will be kept in the file: `<directory_name>/GlobalPatch`) is atomically incremented and the changelog entry is added to the current file, associated to that patch number. Also, a small entry is added to a file `<directory_name>/GlobalChangeLog` which points to the current file. This allows inspecting all changes sequentially by visiting all the files where the changes were made (see C-c C-n). This is obviously useful when maintaining a single thread of version and patch numbers for a set of files.

`off` Turns off version control: C-x C-s then simply saves the file as usual.

Some useful tips:

- If a changelog entry is in fact introduced, the cursor is left at the point in the file where the comment was inserted and the mark is left at the original file point. This allows inspecting (and possibly modifying) the changelog entry, and then returning to the original point in the file by simply typing C-x C-x.
- The first changelog entry is entered by default at the end of the buffer. Later, the changelog entries can be moved anywhere else in the file. New changelog entries are always inserted just above the first changelog entry which appears in the file.
- The comments in changelog entries can be edited at any time.
- If a changelog entry is moved to another file, and version numbers are shared by several files through a directory, the corresponding file pointer in the `<directory_name>/GlobalChangeLog` file needs to be changed also, for the entry to be locatable later using C-c C-n.

⟨C-x⟩ ⟨C-s⟩ This is the standard `emacs` command that saves a buffer by writing the contents into the associated `.pl` file. However, in the Ciao mode, if version control is set to on for ths file, then this command will ask the user before saving whether to introduce a changelog entry documenting the changes performed.

In addition, if:

- the buffer does not already contain a comment specifying the type of version control to be performed,
- and the customizable variable `ciao-ask-for-version-maintenance-type` is set to `yes` (go to the Ciao options menu, LPdoc area to change this, which is by default set to `no`),

then, before saving a buffer, the user will be also automatically asked to choose which kind of version control is desired for the file, as in C-c C-a.

⟨C-c⟩ ⟨C-s⟩ Same as C-x C-s except that it forces prompting for inclusion of a changelog entry even if the buffer is unmodified.

⟨C-c⟩ ⟨n⟩ Force a move to a new major/minor version number (the user will be prompted for the new numbers). Only applicable if using directory-based version maintenance. Note that otherwise it suffices with introducing a changelog entry in the file and changing its version number by hand.

⟨C-c⟩ ⟨C-n⟩ When a unique version numbering is being maintained across several files, this command allows inspecting all changes sequentially by visiting all the files in which the changes were made:

- If in a source file, find the next changelog entry in the source file, open in another window the corresponding `GlobalChangeLog` file, and position the cursor at the corresponding entry. This allows browsing the previous and following changes made, which may perhaps reside in other files in the system.
- If in a `GlobalChangeLog` file, look for the next entry in the file, and open in another window the source file in which the corresponding comment resides, positioning the corresponding comment at the top of the screen. This allows going through a section of the `GlobalChangeLog` file checking all the corresponding comments in the different files in which they occur.

10.13 Generating program documentation

These commands provide some bindings and facilities for generating and viewing the documentation corresponding to the current buffer. The documentation is generated in a temporary directory, which is created automatically. This is quite useful while modifying the documentation for a file, in order to check the output that will be produced, whithout having to set up a documentation directory by hand or to regenerate a large manual of which the file may be a part.

⟨C-c⟩ ⟨D⟩ ⟨B⟩ Generate the documentation for the current buffer in the default format. This allows generating a simple document for the current buffer. Basically, it creates a simple, default `SETTINGS.pl` file, sets `mainfile` in `SETTINGS.pl` to the current buffer and then generates the documentation in a temporary directory. This is useful for seeing how the documentation of a file will format. Note that for generating manuals the best approach is to set up a permanent documentation directory with the appropriate `SETTINGS.pl` file (see the LPdoc manual).

⟨C-c⟩ ⟨D⟩ ⟨F⟩ Change the default output format used by the LPdoc auto-documenter. It is set by default to `html` or to the environment variable `LPDOCFORMAT` if it is defined.

⟨C-c⟩ ⟨D⟩ ⟨S⟩ Visit, or create, the default `SETTINGS.pl` file (which controls all auto-documenter options).

⟨C-c⟩ ⟨D⟩ ⟨G⟩

Generate the documentation according to `SETTINGS.pl` in the default format. This allows generating complex documents but it assumes that `SETTINGS.pl` exists and that the options that it contains (main file, component files, paths, etc.) have been set properly. Documentation is generated in a temporary directory. Note however that for generating complex manuals the best approach is to set up a permanent documentation directory with the appropriate `SETTINGS.pl` and `Makefile` files (see the LPdoc manual).

⟨C-c⟩ ⟨D⟩ ⟨V⟩

Start a viewer on the documentation for the current buffer in the default format.

⟨C-c⟩ ⟨D⟩ ⟨W⟩
: Change the root working dir used by the LPdoc auto-documenter. It is set by default to a new dir under `/tmp` or to the environment variable `LPDOCWDIR` if it is defined.

10.14 Setting top level preprocessor and documenter executables

These commands allow *changing the executables used* when starting the top-level, the preprocessor, or the auto-documenter. They also allow changing the arguments that these executables take, and changing the path where the libraries reside. In the case of the top-level and preprocessor, this should be done only by users which understand the implications, but it is very useful if several versions of Ciao or the preprocessor are available in the system. All these settings can be changed through the *customize* options in the help menu (see Section 10.20 [Customization], page 80).

⟨C-c⟩ ⟨S⟩ ⟨A⟩

⟨C-c⟩ ⟨S⟩ ⟨C⟩ Change the Ciao executable used to run the top level. It is set by default to `ciao` or, to the environment variable `CIAO` if it is defined.

⟨C-c⟩ ⟨S⟩ ⟨C-c⟩
: Change the arguments passed to the Ciao executable. They are set by default to none or, to the environment variable `CIAOARGS` if it is defined.

⟨C-c⟩ ⟨S⟩ ⟨P⟩ Change the executable used to run the Ciao Preprocessor toplevel. It is set by default to `ciaopp` or, to the environment variable `CIAOPP` if it is defined.

⟨C-c⟩ ⟨S⟩ ⟨C-p⟩
: Change the arguments passed to the Ciao preprocessor executable. They are set by default to none or to the environment variable `CIAOPPARGS` if it is defined.

⟨C-c⟩ ⟨S⟩ ⟨L⟩ Change the location of the Ciao library paths (changes the environment variable `CIAOLIB`).

⟨C-c⟩ ⟨S⟩ ⟨D⟩ Change the executable used to run the LPdoc auto-documenter. It is set by default to `lpdoc` or to the environment variable `LPDOC` if it is defined.

⟨C-c⟩ ⟨S⟩ ⟨C-d⟩
: Change the arguments passed to the LPdoc auto-documenter. They are set by default to none or to the environment variable `LPDOCARGS` if it is defined.

⟨C-c⟩ ⟨S⟩ ⟨C-l⟩
: Change the path in which the LPdoc library is installed. It is set by default to `/home/clip/lib` or to the environment variable `LPDOCLIB` if it is defined.

10.15 Other commands

Some other commands which are active in the Ciao mode:

⟨C-c⟩ ⟨C-l⟩ Recenter the most recently used Ciao inferior process buffer (top level or preprocessor).

10.16 Traditional Prolog Mode Commands

These commands provide some bindings and facilities for loading programs, which are present in emacs Prolog modes of traditional Prolog systems (e.g., SICStus). This is useful mainly if the Ciao emacs mode is used with such Prolog systems. Note that these commands (`compile/1` and `consult/1`) are deprecated in Ciao (due to the more advanced, separate compilation model in Ciao) and their use in the Ciao top-level is not recommended.

⟨C-c⟩ ⟨K⟩ Compile the entire buffer.

⟨C-c⟩ ⟨k⟩ Compile a given region.

⟨C-c⟩ ⟨C-k⟩ Compile the predicate around point.

⟨C-c⟩ ⟨C⟩ Consult the entire buffer.

⟨C-c⟩ ⟨c⟩ Consult a given region.

⟨C-c⟩ ⟨C-c⟩ Consult the predicate around point.

10.17 Coexistence with other Prolog-like interfaces

As mentioned previously, the Ciao `emacs` interface can also be used to work with traditional Prolog or CLP systems. Also, the Ciao `emacs` interface (*mode*) can coexist with other Prolog-related `emacs` interfaces (*modes*) (such as, e.g., the `SICStus` Prolog interface). Only one of the interfaces can be active at a time for a given buffer (i.e., for each given file opened inside `emacs`). In order the change a buffer to a given interface, move the cursor to that buffer and type `M-x ...-mode` (e.g., for the Ciao mode, `M-x ciao-mode`).

If several Prolog-related `emacs` interfaces are loaded, then typically the *last* one to be loaded takes precedence, in the sense that this will be the interface in which `emacs` will be set when opening files which have a `.pl` ending (this depends a bit on how things are set up in your `.emacs` file).

10.18 Getting the Ciao mode version

⟨C-c⟩ ⟨v⟩ Report the version of the emacs Ciao mode.

10.19 Using Ciao mode capabilities in standard shells

The capabilities (commands, coloring, error location, ...) which are active in the Ciao *inferior* mode can also be made available in any standard command line shell which is being run within emacs. This can be enabled by going to the buffer in which the shell is running and typing "⟨M-x⟩ `ciao-inferior-mode`". This is very useful for example when running the stand-alone compiler, the `lpdoc` auto-documenter, or even certain user applications (those that use the standard error message library) in an emacs sub-shell. Turning the Ciao inferior mode on on that sub-shell will highlight and color the error messages, and automatically find and visit the locations in the files in which the errors are reported.

Finally, one the most useful applications of this is when using the embedded debugger (a version of the debugger which can be embedded into executables so that an interactive debugging session can be triggered at any time while running that executable without needing the top-level shell). If an application is run in a shell buffer which has been set with Ciao inferior mode (⟨M-x⟩ `ciao-inferior-mode`) and this application starts emitting output from the embedded debugger (i.e., which contains the embedded debugger and is debugging its code) then the Ciao emacs mode will be able to follow these messages, for example tracking execution in the source level code. This also works if the application is written in a combination of languages, provided the

parts written in Ciao are compiled with the embedded debugger package and is thus a covenient way of debugging multi-language applications. The only thing needed is to make sure that the output messages appear in a shell buffer that is in Ciao inferior mode.

10.20 Customization

This section explains all variables used in the Ciao emacs mode which can be customized by users. Such customization can be performed (in later versions of `emacs`) from the `emacs` menus (`Help -> Customize -> Top-level Customization Group`), or also by adding a `setq` expression in the `.emacs` file. Such `setq` expression should be similar to:

```
(setq <variable> <new_value>)
```

The following sections list the different variables which can be customized for `ciao`, `ciaopp` and `lpdoc`.

10.20.1 Ciao general variables

`ciao-ask-for-version-maintenance-type` (*string*)
: If turned to yes the system asks prompts to set version control when saving files that do not set a version control system explicitly within the file.

`ciao-clip-logo` (*file*)
: CLIP logo image.

`ciao-create-sample-file-on-startup` (*boolean*)
: When starting the Ciao environment using ciao-startup two buffers are opened: one with a toplevel and another with a sample file. This toggle controls whether the sample file, meant for novice users, is created or not. Set by default, non-novice users will probably want to turn it off.

`ciao-first-indent-width` (*integer*)
: First level indentation for a new goal.

`ciao-indent-width` (*integer*)
: Indentation for a new goal.

`ciao-inhibit-toolbar` (*boolean*)
: *Non-nil means don't use the specialized Ciao toolbar.

`ciao-library-path` (*string*)
: Path to the Ciao System libraries (reads/sets the CIAOLIB environment variable). Typically left empty, since ciao executables know which library to use.

`ciao-locate-also-note-messages` (*boolean*)
: If set, also when errors of type NOTE are detected the corresponding file is visited and the location marked. It is set to nil by default because sometimes the user prefers not to take any action with respect to these messages (for example, many come from the documenter, indicating that adding certain declarations the documentation would be improved).

`ciao-locate-errors-after-run` (*boolean*)
: If set, location of any errors produced when running Ciao tools (loading or preprocessing code, running the documenter, etc.) will be initiated automatically. I.e., after running a command, the system will automatically highlight any error messages and the corresponding areas in source files if possible. If set to nil this location will only happen after typing C-c ‘ or accessing the corresponding menu or tool bar button.

`ciao-logo` (*file*)
: Ciao logo image.

`ciao-main-filename` (*string*)
: Name of main file in a multiple module program. Setting thsi is very useful when working on a multi-module program because it allows issuing a load command after working on an inferior module which will reload from the main module, thus also reloading automatically all dependent modules.

`ciao-os-shell-prompt-pattern` (*string*)
: Regular expression used to describe the shell prompt pattern, so that error location works in inferior shells. This is useful for example so that errors are located when generating documentation, and also when using the embedded debugger or any other application in a shell. It is best to be as precise as possible when defining this so that the standard Ciao error location does not get confused.

`ciao-query` (*string*)
: Query to use in Ciao. Setting this is useful when using a long or complicated query because it saves from having to type it over and over again. It is possible to set that this query will be issued any time a program is (re)loaded.

`ciao-system` (*string*)
: Name of Ciao executable which runs the classical top level.

`ciao-system-args` (*string*)
: Arguments passed to Ciao toplevel executable.

`ciao-toplevel-buffer-name` (*string*)
: Basic name of the buffer running the Ciao toplevel inferior process.

`ciao-user-directives` (*list*)
: List of identifiers of any directives defined by users which you would like highlighted (colored). Be careful, since wrong entries may affect other syntax highlighting.

10.20.2 CiaoPP variables

`ciao-ciaopp-buffer-name` (*string*)
: Basic name of the buffer running the Ciao preprocessor inferior process.

`ciao-ciaopp-gmenu-buffer-name` (*string*)
: Basic name of the buffer running the Ciao preprocessor graphical menu interface.

`ciao-ciaopp-system` (*string*)
: Name of Ciao preprocessor executable.

`ciao-ciaopp-system-args` (*string*)
: Arguments passed to Ciao preprocessor executable.

`ciao-ciaopp-use-graphical-menu` (*boolean*)
: If set, an interactive graphical menu is used for controlling CiaoPP, instead of asking ascii questions in the CiaoPP buffer.

10.20.3 LPdoc variables

`ciao-lpdoc-buffer-name` (*string*)
: Basic name of the buffer running the auto-documenter inferior process.

`ciao-lpdoc-docformat` (*symbol*)
: Name of default output format used by LPdoc.

`ciao-lpdoc-libpath` (*directory*)
: Path in which the LPdoc library is installed.

`ciao-lpdoc-system` (*string*)
: Name of LPdoc auto-documenter executable.

`ciao-lpdoc-system-args` (*string*)
: Arguments passed to LPdoc executable.

`ciao-lpdoc-wdir-root` (*directory*)
: Name of root working dir used by LPdoc.

10.20.4 Faces used in syntax-based highlighting (coloring)

`ciao-button-pressed-widget-face` (*face*)
: Face used for documentation text.

`ciao-button-widget-face` (*face*)
: Face used for documentation text.

`ciao-edit-widget-face` (*face*)
: Face used for documentation text.

`ciao-face-answer-val` (*face*)
: Face to use for answer values in top level.

`ciao-face-answer-var` (*face*)
: Face to use for answer variables in top level.

`ciao-face-builtin-directive` (*face*)
: Face to use for the standard directives.

`ciao-face-check-assrt` (*face*)
: Face to use for check assertions.

`ciao-face-checked-assrt` (*face*)
: Face to use for checked assertions.

`ciao-face-ciaopp-option` (*face*)
: Face to use for CiaoPP option menus.

`ciao-face-clauseheadname` (*face*)
: Face to use for clause head functors.

`ciao-face-comment` (*face*)
: Face to use for code comments using fixed pitch (double %).

`ciao-face-comment-variable-pitch` (*face*)
: Face to use for code comments using variable pitch (single %).

`ciao-face-concurrency-op` (*face*)
: Face to use for concurrency operators.

`ciao-face-condcode-directive` (*face*)
: Face to use for the conditional code directives.

`ciao-face-cut` (*face*)
: Face to use for cuts.

`ciao-face-debug-breakpoint` (*face*)
: Face to use with breakpoints in source debugger.

`ciao-face-debug-call` (*face*)
: Face to use when at call port in source debugger.

`ciao-face-debug-exit` (*face*)
: Face to use when at exit port in source debugger.

`ciao-face-debug-expansion` (*face*)
: Face to use in source debugger when source literal not located.

`ciao-face-debug-fail` (*face*)
: Face to use when at fail port in source debugger.

`ciao-face-debug-mess` (*face*)
: Face to use for debug messages.

`ciao-face-debug-redo` (*face*)
: Face to use when at redo port in source debugger.

`ciao-face-entry-assrt` (*face*)
: Face to use for entry assertions.

`ciao-face-error-mess` (*face*)
: Face to use for error messages.

`ciao-face-false-assrt` (*face*)
: Face to use for false assertions.

`ciao-face-fontify-sectioning` (*symbol*)
: Whether to fontify sectioning macros with varying height or a color face.

  If it is a number, use varying height faces. The number is used for scaling starting from 'ciao-face-sectioning-5-face'. Typically values from 1.05 to 1.3 give best results, depending on your font setup. If it is the symbol 'color', use 'font-lock-type-face'.

  Caveats: Customizing the scaling factor applies to all sectioning faces unless those face have been saved by customize. Setting this variable directly does not take effect unless you call 'ciao-face-update-sectioning-faces' or restart Emacs.

  Switching from 'color' to a number or vice versa does not take effect unless you call M-x font-lock-fontify-buffer or restart Emacs.

`ciao-face-funexp-atom` (*face*)
: Face to use for atoms in functional notation.

`ciao-face-highlight-code` (*face*)
: Face to use for highlighting code areas (e.g., when locating the code area that an error message refers to).

`ciao-face-library-directive` (*face*)
: Face to use for directives defined in the library.

`ciao-face-lpdoc-bug-comment` (*face*)
: Face to use for LPdoc bug comments.

`ciao-face-lpdoc-command` (*face*)
: Face to use LPdoc commands inserted in documentation text.

`ciao-face-lpdoc-comment` (*face*)
: Face to use for LPdoc textual comments.

`ciao-face-lpdoc-comment-variable-pitch` (*face*)
: Face to use for LPdoc textual comments in variable pitch.

`ciao-face-lpdoc-crossref` (*face*)
: Face to use for LPdoc cross-references.

`ciao-face-lpdoc-include` (*face*)
: Face to use for LPdoc include commands.

`ciao-face-lpdoc-verbatim` (*face*)
: Face to use for LPdoc verbatim text.

`ciao-face-lpdoc-version-comment` (*face*)
: Face to use for LPdoc version comments.

`ciao-face-modedef-assrt` (*face*)
: Face to use for modedef definitions.

`ciao-face-module-directive` (*face*)
: Face to use for the module-related directives.

`ciao-face-no-answer` (*face*)
: Face to use for no answer in top level.

`ciao-face-note-mess` (*face*)
: Face to use for note messages.

`ciao-face-other-mess` (*face*)
: Face to use for other messages.

`ciao-face-predicate-directive` (*face*)
: Face to use for the predicate-related directives.

`ciao-face-prompt` (*face*)
: Face to use for prompts in top-level and shells.

`ciao-face-prop-assrt` (*face*)
: Face to use for property definitions.

`ciao-face-quoted-atom` (*face*)
: Face to use for quoted atoms.

`ciao-face-script-header` (*face*)
: Face to use for script headers.

`ciao-face-sectioning-0-face` (*face*)
: Face for sectioning commands at level 0.

  Probably you don't want to customize this face directly. Better change the base face 'ciao-face-sectioning-5-face' or customize the variable 'ciao-face-fontify-sectioning'.

`ciao-face-sectioning-1-face` (*face*)
: Face for sectioning commands at level 1.

  Probably you don't want to customize this face directly. Better change the base face 'ciao-face-sectioning-5-face' or customize the variable 'ciao-face-fontify-sectioning'.

`ciao-face-sectioning-2-face` (*face*)
: Face for sectioning commands at level 2.

  Probably you don't want to customize this face directly. Better change the base face 'ciao-face-sectioning-5-face' or customize the variable 'ciao-face-fontify-sectioning'.

`ciao-face-sectioning-3-face` (*face*)
: Face for sectioning commands at level 3.

  Probably you don't want to customize this face directly. Better change the base face 'ciao-face-sectioning-5-face' or customize the variable 'ciao-face-fontify-sectioning'.

`ciao-face-sectioning-4-face` (*face*)
: Face for sectioning commands at level 4.

  Probably you don't want to customize this face directly. Better change the base face 'ciao-face-sectioning-5-face' or customize the variable 'ciao-face-fontify-sectioning'.

`ciao-face-sectioning-5-face` (*face*)
: Face for sectioning commands at level 5.

`ciao-face-startup-mess` (*face*)
: Face to use for system splash message.

`ciao-face-string` (*face*)
: Face to use for strings.

`ciao-face-test-assrt` (*face*)
: Face to use for test assertions.

`ciao-face-texec-assrt` (*face*)
: Face to use for texec assertions.

`ciao-face-true-assrt` (*face*)
: Face to use for true assertions.

`ciao-face-trust-assrt` (*face*)
: Face to use for trust assertions.

`ciao-face-type-assrt` (*face*)
: Face to use for type definitions.

`ciao-face-user-directive` (*face*)
: Face to use for directives defined by the user (see ciao-user-directives custom variable to add new ones).

`ciao-face-variable` (*face*)
: Face to use for variables.

`ciao-face-warning-mess` (*face*)
: Face to use for warning messages.

`ciao-face-yes-answer` (*face*)
: Face to use for yes answer in top level.

`ciao-faces-use-variable-pitch-in-comments` (*boolean*)
: Controls whether variable pitch fonts are used when highlighting comments. Unset by default. After changing this you must exit and reinitialize for the change to take effect.

`ciao-menu-error-widget-face` (*face*)
: Face used for menu error representation in graphical interface.

`ciao-menu-note-widget-face` (*face*)
: Face used for menu note representation in graphical interface.

`ciao-text-widget-face` (*face*)
: Face used for documentation text.

10.21 Installation of the Ciao emacs interface

If opening a file ending with `.pl` puts emacs in another mode (such as `perl` mode, which is the –arguably incorrect– default setting in some `emacs` distributions), then either the emacs mode was not installed or the installation settings are being overwritten by other settings in your `.emacs` file or in some library. In any case, you can set things manually so that the Ciao mode is loaded by default in your system. This can be done by including in your `.emacs` file a line such as:

```
(load <CIAOLIBDIR>/ciao-mode-init)
```

This loads the above mentioned file from the Ciao library, which contains the following lines (except that the paths are changed during installation to appropriate values for your system):

```
; -*- mode: emacs-lisp; -*-
;;
;; Ciao/Prolog mode initialization
;; -------------------------------
;; (can normally be used with other Prolog modes and the default prolog.el)
;;
(setq load-path (cons <v>CIAOLIBDIR</v> load-path))
;; Java mode in ciao
(setq load-path
   (cons <v>CIAOLIBDIR</v> load-path))
(defun load-java-ciaopp-mode ()
  (require 'java-ciaopp)
  (java-ciaopp-setup))
(add-hook 'java-mode-hook 'load-java-ciaopp-mode)

(autoload 'run-ciao-toplevel "ciao"
          "Start a Ciao/Prolog top-level sub-process." t)
(autoload 'ciao-startup "ciao"
          "The Ciao/Prolog program development system startup screens." t)
(autoload 'ciao "ciao"
          "Start a Ciao/Prolog top-level sub-process." t)
(autoload 'prolog "ciao"
          "Start a Ciao/Prolog top-level sub-process." t)
(autoload 'run-ciao-preprocessor "ciao"
          "Start a Ciao/Prolog preprocessor sub-process." t)
(autoload 'ciaopp "ciao"
          "Start a Ciao/Prolog preprocessor sub-process." t)
(autoload 'ciao-mode "ciao"
          "Major mode for editing and running Ciao/Prolog" t)
(autoload 'ciao-inferior-mode "ciao"
          "Major mode for running Ciao/Prolog, CiaoPP, LPdoc, etc." t)
(setq auto-mode-alist (cons '("\\.pl$" . ciao-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.pls$" . ciao-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.lpdoc$" . ciao-mode) auto-mode-alist))
(setq completion-ignored-extensions
      (append '(".dep" ".itf" ".po" ".asr" ".cpx")
              completion-ignored-extensions))
;; ------------------------------------------------------------------------
;; In Un*x, the following (or similar) lines should be included in your
;; .cshrc or .profile to find the manuals (the Ciao installation leaves
;; in the Ciao library directory 'DOTcshrc' and 'DOTprofile' files with
;; the right paths which can be included directly in your startup scripts):
;;
;; setenv INFOPATH /usr/local/info:/usr/info:<v>LPDOCDIR</v>
;; ------------------------------------------------------------------------
```

If you would like to configure things in a different way, you can also copy the contents of this file to your `.emacs` file and make the appropriate changes. For example, if you do not want `.pl` files to be put automatically in Ciao mode, then comment out (or remove) the line:

`(setq auto-mode-alist ... )`

You will then need to switch manually to Ciao mode by typing `M-x ciao-mode` after opening a Ciao file.

If you are able to open the Ciao menu but the Ciao manuals are not found or the `ciao` command (the top-level) is not found when loading `.pl` files, the probable cause is that you do not have the Ciao paths in the `INFOPATH` and `MANPATH` *environment variables* (whether these variables are set automatically or not for users depends on how the Ciao system was installed). Under Un*x, you can add these paths easily by including the line:

```
source <CIAOLIBDIR>/DOTcshrc
```

in your `.login` or `.cshrc` files if you are using `csh` (or `tcsh`, etc.), or, alternatively, the line:

```
. <CIAOLIBDIR>/DOTprofile
```

in your `.login` or `.profile` files if you are using `sh` (or `bash`, etc.). See the Ciao installation instructions (Chapter 231 [Installing Ciao from the source distribution], page 1101 or Chapter 232 [Installing Ciao from a Win32 binary distribution], page 1111) for details.

10.22 Emacs version compatibility

This mode is currently being developed within `GNU emacs` version 21.3. It should also (hopefully) work with all other 21.XX, 20.XX, and later 19.XX versions. We also try our best to keep things working under `xemacs` and under the diverse emacs native ports for the mac.

10.23 Acknowledgments (ciao.el)

This code is derived from the 1993 version of the emacs interface for &-Prolog by M. Hermenegildo, itself derived from the original `prolog.el` by *Masanobu Umeda* with changes by *Johan Andersson*, *Peter Olin*, *Mats Carlsson*, and *Johan Bevemyr* of *SICS*, Sweden. Other changes also by Daniel Cabeza and Manuel C. Rodriguez. See the changelog for details.

PART II - The Ciao basic language (engine)

Author(s): The CLIP Group.

This part documents the *Ciao basic builtins*. These predefined predicates and declarations are available in every program, unless the `pure` package is used (by using a `:- module(_,_,[pure]).` declaration or `:- use_package(pure).`). These predicates are contained in the `engine` directory within the `lib` library. The rest of the library predicates, including the packages that provide most of the ISO-Prolog builtins, are documented in subsequent parts.

11 The module system

Author(s): Daniel Cabeza, The CLIP Group.

Modularity is a basic notion in a modern computer language. Modules allow dividing programs in several parts, which have its own independent name spaces. The module system in Ciao [CH00a], as in many other Prolog implementations, is procedure based. This means that predicate names are local to a module, but functor/atom names in data are shared (at least by default).

The predicates visible in a module are the predicates defined in that module, plus the predicates imported from other modules. Only predicates exported by a module can be imported from other modules. The default module of a given predicate name is the local one if the predicate is defined locally, else the last module from which the predicate is imported, where explicit imports have priority over implicit ones (that is, a predicate imported through a `use_module/2` declaration is always preferred over a predicate imported through a `use_module/1` declaration). To refer to a predicate from a module which is not the default module for that predicate the name has to be module qualified. A module qualified predicate name has the form `Module:Predicate` as in the call `debugger:debug_module(M)`. Note that in Ciao this module qualification cannot be used for gaining access to predicates that have not been imported, nor for defining clauses of other modules.

All predicates defined in files with no module declaration belong to a special module called `user`, from which they are all implicitly exported. This provides backward compatibility for programs written for implementations with no module system and allows dividing programs into several files without being aware of the module system at all. Note that this feature is only supported for the above-mentioned backward-compatibility reasons, and the use of `user` files is discouraged. Many attractive compilation features of Ciao cannot be supported for `user` modules.

The case of multifile predicates (defined with the declaration `multifile/1`) is also special. Multifile predicates can be defined by clauses distributed in several modules, and all modules which define a predicate as multifile can use that predicate. The name space of multifile predicates is independent, as if they belonged to the special module `multifile`.

Every `user` or module file imports implicitly a number of modules called builtin modules. They are imported before all other importations of the module, thus allowing the redefinition of any of their predicates (with the exception of `true/0`) by defining local versions or importing them from other modules. Importing explicitly from a builtin module, however, disables the implicit importation of the rest (this feature is used by package `library(pure)` to define pure Prolog code).

11.1 Usage and interface (`modules`)

- **Library usage:**
  Modules are an intrinsic feature of Ciao, so nothing special has to be done to use them.

11.2 Documentation on internals (`modules`)

module/3: DECLARATION

Usage: `:- module(Name,Exports,Packages).`

– *Description:* Declares a module of name `Name` which exports the predicates in `Exports`, and uses the packages in `Packages`. `Name` must match with the name of the file where the module resides, without extension. For each source in `Packages`, a package file is used. If the source is specified with a path alias, this is the file included, if it is an atom, the library paths are searched. See `package/1` for a brief description of package files.

This directive must appear the first in the file.

Also, if the compiler finds an unknown declaration as the first term in a file, the name of the declaration is regarded as a package library to be included, and the arguments of the declaration (if present) are interpreted like the arguments of `module/3`.

– *The following properties hold at call time:*

`Name` is a module name (an atom). (modules:modulename/1)

`Exports` is a list of `prednames`. (basic_props:list/2)

`Packages` is a list of `sourcenames`. (basic_props:list/2)

module/2: DECLARATION

Usage: :- `module(Name,Exports).`

– *Description:* Same as directive `module/3`, with an implicit package `default`. This default package provides all the standard features provided by most Prolog systems so that Prolog programs with traditional `module/2` declarations can run without any change.

– *The following properties hold at call time:*

`Name` is a module name (an atom). (modules:modulename/1)

`Exports` is a list of `prednames`. (basic_props:list/2)

package/1: DECLARATION

Usage: :- `package(Name).`

– *Description:* Declares a package of name `Name`. Like in modules, `Name` must match with the name of the file where the package resides, without extension. This directive must appear the first in the file.

Package files provide syntactic extensions and their related functionalities by defining operators, new declarations, code translations, etc., as well as declaring imports from other modules and defining additional code. Most Ciao syntactic and semantic extensions, such as functional syntax, constraint solving, or breadth-first search are implemented as packages.

– *The following properties hold at call time:*

`Name` is a module name (an atom). (modules:modulename/1)

export/1: DECLARATION

Usage 1: :- `export(Pred).`

– *Description:* Adds `Pred` to the set of exported predicates.

– *The following properties hold at call time:*

`Pred` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

Usage 2: :- `export(Exports)`.

- *Description:* Adds `Exports` to the set of exported predicates.
- *The following properties hold at call time:*

  `Exports` is a list of `prednames`. (basic_props:list/2)

use_module/2: DECLARATION

Usage: :- `use_module(Module,Imports)`.

- *Description:* Specifies that this code imports from the module defined in `Module` the predicates in `Imports`. The imported predicates must be exported by the other module.
- *The following properties hold at call time:*

  `Module` is a source name. (streams_basic:sourcename/1)

  `Imports` is a list of `prednames`. (basic_props:list/2)

use_module/1: DECLARATION

Usage: :- `use_module(Module)`.

- *Description:* Specifies that this code imports from the module defined in `Module` all the predicates exported by it. The previous version with the explicit import list is preferred to this as it minimizes the chances to have to recompile this code if the other module changes.
- *The following properties hold at call time:*

  `Module` is a source name. (streams_basic:sourcename/1)

import/2: DECLARATION

Usage: :- `import(Module,Imports)`.

- *Description:* Declares that this code imports from the module with name `Module` the predicates in `Imports`.

  Important note: this declaration is intended to be used when the current module or the imported module is going to be dynamically loaded, and so the compiler does not include the code of the imported module in the current executable (if only because the compiler cannot know the location of the module file at the time of compilation). For the same reason the predicates imported are not checked to be exported by `Module`. Its use in other cases is strongly discouraged, as it disallows many compiler optimizations.
- *The following properties hold at call time:*

  `Module` is a module name (an atom). (modules:modulename/1)

  `Imports` is a list of `prednames`. (basic_props:list/2)

reexport/2: DECLARATION

Usage: :- `reexport(Module,Preds)`.

– *Description:* Specifies that this code reexports from the module defined in `Module` the predicates in `Preds`. This implies that this module imports from the module defined in `Module` the predicates in `Preds`, an also that this module exports the predicates in `Preds` .

– *The following properties hold at call time:*

`Module` is a source name. (streams_basic:sourcename/1)

`Preds` is a list of `predname`s. (basic_props:list/2)

reexport/1: DECLARATION

Usage: :- `reexport(Module)`.

– *Description:* Specifies that this code reexports from the module defined in `Module` all the predicates exported by it. This implies that this module imports from the module defined in `Module` all the predicates exported by it, an also that this module exports all such predicates .

– *The following properties hold at call time:*

`Module` is a source name. (streams_basic:sourcename/1)

meta_predicate/1: DECLARATION

Usage: :- `meta_predicate MetaSpecs`.

– *Description:* Specifies that the predicates in `MetaSpecs` have arguments which has to be module expanded (predicates, goals, etc). The directive is only mandatory for exported predicates (in modules). This directive is defined as a prefix operator in the compiler.

– *The following properties hold at call time:*

`MetaSpecs` is a sequence of `metaspec`s. (basic_props:sequence/2)

modulename/1: REGTYPE

A module name is an atom, not containing characters ':' or '$'. Also, `user` and `multifile` are reserved, as well as the module names of all builtin modules (because in an executable all modules must have distinct names).

Usage: `modulename(M)`

– *Description:* `M` is a module name (an atom).

metaspec/1: REGTYPE

A meta-predicate specification for a predicate is the functor of that predicate applied to terms which represent the kind of module expansion that should be applied to each argument. Possible contents are represented as:

`?,+,-,_` These values denote that this argument is not module expanded.

`goal` This argument will be a term denoting a goal (either a simple or complex one) which will be called. For commpatibility reasons it can be named as `:` as well.

`clause` This argument will be a term denoting a clause.

`fact` This argument should be instantiated to a term denoting a fact (head-only clause).

`spec` This argument should be instantiated to a predicate name, as Functor/Arity.

`pred(`*N*`)` This argument should be instantiated to a predicate construct to be called by means of a `call/`*N* predicate call (see `call/2`).

`list(`*Meta*`)`
This argument should be instantiated to a list of terms as described by *Meta* (e.g. `list(goal)`).

`addterm(Meta)`
This argument should be instantiated to the meta-data specified by *Meta*, and an argument added after this one will carry the original data without module expansion. Not intended to be used by normal users.

`addmodule(Meta)`
This argument should be instantiated to the meta-data specified by *Meta*, and in an argument added after this one will be passed the calling module, for example to allow handling more involved meta-data by using conversion builtins. `addmodule` is an alias of `addmodule(?)`. Not intended to be used by normal users.

Usage: `metaspec(M)`

– *Description:* `M` is a meta-predicate specification.

12 Directives for using code in other files

Author(s): Daniel Cabeza.

Documentation for the directives used to load code into Ciao Prolog (both from the toplevel shell and by other modules).

12.1 Usage and interface (`loading_code`)

- **Library usage:**
  These directives are builtin in Ciao, so nothing special has to be done to use them.

12.2 Documentation on internals (`loading_code`)

ensure_loaded/1: DECLARATION

Usage: :- `ensure_loaded(File)`. ⟨• ISO •⟩

– *Description:* Specifies that the code present in `File` will be included in the executable being prepared, in the `user` module. The file `File` cannot have a module declaration. This directive is intended to be used by programs not divided in modules. Dividing programs into modules is however strongly encouraged, since most of the attractive features of Ciao (such as static debugging and global optimization) are only partially available for `user` modules.

– *The following properties should hold at call time:*
`File` is a source name. (streams_basic:sourcename/1)

include/1: DECLARATION

Usage: :- `include(File)`. ⟨• ISO •⟩

– *Description:* The contents of the file `File` are included in the current program text exactly as if they had been written in place of this directive.

– *The following properties should hold at call time:*
`File` is a source name. (streams_basic:sourcename/1)

use_package/1: DECLARATION

:- `use_package(Package)`.

Specifies the use in this file of the packages defined in `Package`. See the description of the third argument of `module/3` for an explanation of package files.

This directive must appear the first in the file, or just after a `module/3` declaration. A file with no module declaration, in the absence of this directive, uses an implicit package `default` (see Chapter 31 [Other predicates and features defined by default], page 211).

Usage 1: :- `use_package(Package)`.

– *The following properties should hold at call time:*
`Package` is a source name. (streams_basic:sourcename/1)

Usage 2: :- `use_package(Package)`.

– *The following properties should hold at call time:*
`Package` is a list of `sourcename`s. (basic_props:list/2)

13 Control constructs/predicates

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This module contains the set of basic control predicates, except the predicates dealing with exceptions, which are in Chapter 23 [Exception and Signal handling], page 177.

13.1 Usage and interface (`basiccontrol`)

- **Library usage:**
  These predicates/constructs are builtin in Ciao, so nothing special has to be done to use them. In fact, as they are hardwired in some parts of the system, most of them cannot be redefined.
- **Exports:**
  - *Predicates:*
    `,/2`, `;/2`, `->/2`, `!/0`, `\+/1`, `if/3`, `true/0`, `fail/0`, `repeat/0`, `false/0`, `otherwise/0`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`, `debugger/debugger`.

13.2 Documentation on exports (`basiccontrol`)

,/2: PREDICATE

`P,Q`

Conjunction (`P` *and* `Q`).

Meta-predicate with arguments: `goal,goal`.

Usage: ⟨• ISO •⟩

- *The following properties should hold at call time:*

  `P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

  `Q` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

;/2: PREDICATE

`P;Q`

Disjunction (`P` *or* `Q`). Note that in Ciao `|/2` is not equivalent to `;/2`.

Meta-predicate with arguments: `goal;goal`.

Usage: ⟨• ISO •⟩

- *The following properties should hold at call time:*

  `P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

  `Q` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

->/2: PREDICATE

`P->Q`

If `P` then `Q` else fail, using first solution of `P` only. Also, `(P -> Q ; R)`, if `P` then `Q` else `R`, using first solution of `P` only. No cuts are allowed in `P`.

Meta-predicate with arguments: `goal->goal`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*

`P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Q` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

!/0: PREDICATE

Usage: ⟨• ISO •⟩

– *Description:* Commit to any choices taken in the current predicate.

– *The following properties hold globally:*

All calls of the form `!` are deterministic. (native_props:is_det/1)

All the calls of the form `!` do not fail. (native_props:not_fails/1)

Goal `!` produces `1` solutions. (native_props:relations/2)

\+/1: PREDICATE

`\+P`

Goal `P` is not provable (negation by failure). Fails if `P` has a solution, and succeeds otherwise. No cuts are allowed in `P`.

Meta-predicate with arguments: `\+goal`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*

`P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP as `not(X)`. (basic_props:native/2)

All calls of the form `\+P` are deterministic. (native_props:is_det/1)

if/3: PREDICATE

`if(P,Q,R)`

If `P` then `Q` else `R`, exploring all solutions of `P`. No cuts are allowed in `P`.

Meta-predicate with arguments: `if(goal,goal,goal)`.

Usage: `if(A,B,C)`

– *The following properties should hold at call time:*

`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`C` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`B` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`C` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties hold upon exit:*

`A` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`B` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`C` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

true/0: PREDICATE

General properties:

– *The following properties hold globally:*

`true` is evaluable at compile-time. (basic_props:eval/1)

Usage: ⟨• ISO •⟩

– *Description:* Succeed (noop).

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`true` is side-effect `free`. (basic_props:sideff/2)

All calls of the form `true` are deterministic. (native_props:is_det/1)

All the calls of the form `true` do not fail. (native_props:not_fails/1)

Goal `true` produces `1` solutions. (native_props:relations/2)

fail/0: PREDICATE

General properties:

– *The following properties hold globally:*

`fail` is evaluable at compile-time. (basic_props:eval/1)

– *The following properties hold globally:*

`fail` is equivalent to `fail`. (basic_props:equiv/2)

Usage: ⟨• ISO •⟩

– *Description:* Fail, backtrack immediately.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`fail` is side-effect `free`. (basic_props:sideff/2)

All calls of the form `fail` are deterministic. (native_props:is_det/1)

Calls of the form `fail` fail. (native_props:fails/1)

Goal `fail` produces `0` solutions. (native_props:relations/2)

repeat/0: PREDICATE

Usage: ⟨• ISO •⟩

– *Description:* Generates an infinite sequence of backtracking choices.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`repeat` is side-effect `free`. (basic_props:sideff/2)

false/0: PREDICATE

General properties:

– *The following properties hold globally:*

`false` is side-effect `free`. (basic_props:sideff/2)

`false` is evaluable at compile-time. (basic_props:eval/1)

otherwise/0: PREDICATE

General properties:

– *The following properties hold globally:*

`otherwise` is side-effect `free`. (basic_props:sideff/2)

`otherwise` is evaluable at compile-time. (basic_props:eval/1)

13.3 Known bugs and planned improvements (`basiccontrol`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

14 Basic builtin directives

Author(s): Daniel Cabeza.

This chapter documents the basic builtin directives in Ciao, additional to the documented in other chapters. These directives are natively interpreted by the Ciao compiler (`ciaoc`).

Unlike in other Prolog systems, directives in Ciao are not goals to be *executed* by the compiler or top level. Instead, they are *read* and acted upon by these programs. The advantage of this is that the effect of the directives is consistent for executables, code loaded in the top level, code analyzed by the preprocessor, etc.

As a result, by default only the builtin directives or declarations defined in this manual can be used in user programs. However, it is possible to define new declarations using the `new_declaration/1` and `new_declaration/2` directives (or using packages including them). Also, packages may define new directives via code translations.

14.1 Usage and interface (`builtin_directives`)

- **Library usage:**
  These directives are builtin in Ciao, so nothing special has to be done to use them.

14.2 Documentation on internals (`builtin_directives`)

multifile/1: DECLARATION

Usage: :- `multifile Predicates`. ⟨• ISO •⟩

– *Description:* Specifies that each predicate in `Predicates` may have clauses in more than one file. Each file that contains clauses for a multifile predicate must contain a directive multifile for the predicate. The directive should precede all clauses of the affected predicates, and also dynamic/data declarations for the predicate. This directive is defined as a prefix operator in the compiler.

– *The following properties should hold at call time:*

`Predicates` is a sequence or list of `prednames`. (basic_props:sequence_or_list/2)

discontiguous/1: DECLARATION

Usage: :- `discontiguous Predicates`. ⟨• ISO •⟩

– *Description:* Specifies that each predicate in `Predicates` may be defined in this file by clauses which are not in consecutive order. Otherwise, a warning is signaled by the compiler when clauses of a predicate are not consecutive (this behavior is controllable by the prolog flag *discontiguous_warnings*). The directive should precede all clauses of the affected predicates. This directive is defined as a prefix operator in the compiler.

– *The following properties should hold at call time:*

`Predicates` is a sequence or list of `prednames`. (basic_props:sequence_or_list/2)

impl_defined/1: DECLARATION

Usage: :- `impl_defined(Predicates)`.

– *Description:* Specifies that each predicate in `Predicates` is *impl*icitly *defined* in the current prolog source, either because it is a builtin predicate or because it is defined in a C file. Otherwise, a warning is signaled by the compiler when an exported predicate is not defined in the module or imported from other module.

– *The following properties should hold at call time:*

`Predicates` is a sequence or list of `prednames`. (basic_props:sequence_or_list/2)

redefining/1: DECLARATION

Usage: :- `redefining(Predicate).`

– *Description:* Specifies that this module redefines predicate `Predicate`, also imported from other module, or imports it from more than one module. This prevents the compiler giving warnings about redefinitions of that predicate. `Predicate` can be partially (or totally) uninstantiated, to allow disabling those warnings for several (or all) predicates at once.

– *The following properties should hold at call time:*

`Predicate` is *compatible* with `predname` (basic_props:compat/2)

initialization/1: DECLARATION

Usage: :- `initialization(Goal).` ⟨• ISO •⟩

– *Description:* `Goal` will be executed at the start of the execution of any program containing the current code. The initialization of a module/file never runs before the initializations of the modules from which the module/file imports (excluding circular dependences).

– *The following properties should hold at call time:*

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

on_abort/1: DECLARATION

Usage: :- `on_abort(Goal).`

– *Description:* `Goal` will be executed after an abort of the execution of any program containing the current code.

– *The following properties should hold at call time:*

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

15 Basic data types and properties

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This library contains the set of basic properties used by the builtin predicates, and which constitute the basic data types and properties of the language. They can be used both as type testing builtins within programs (by calling them explicitly) and as properties in assertions.

15.1 Usage and interface (basic_props)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Properties:*
    `member/2`, `compat/2`, `inst/2`, `iso/1`, `deprecated/1`, `not_further_inst/2`, `sideff/2`, `regtype/1`, `native/1`, `native/2`, `no_rtcheck/1`, `eval/1`, `equiv/2`, `bind_ins/1`, `error_free/1`, `memo/1`, `filter/2`, `pe_type/1`.
  - *Regular Types:*
    `term/1`, `int/1`, `nnegint/1`, `flt/1`, `num/1`, `atm/1`, `struct/1`, `gnd/1`, `gndstr/1`, `constant/1`, `callable/1`, `operator_specifier/1`, `list/1`, `list/2`, `nlist/2`, `sequence/2`, `sequence_or_list/2`, `character_code/1`, `string/1`, `num_code/1`, `predname/1`, `atm_or_atm_list/1`, `flag_values/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`, `terms_check`.

15.2 Documentation on exports (basic_props)

term/1: REGTYPE

The most general type (includes all possible terms).

General properties: `term(X)`

- *The following properties hold globally:*
  `term(X)` is side-effect `free`. (basic_props:sideff/2)

`term(X)`

- *The following properties hold globally:*
  `term(X)` is evaluable at compile-time. (basic_props:eval/1)

`term(X)`

- *The following properties hold globally:*
  `term(X)` is equivalent to `true`. (basic_props:equiv/2)

Usage: `term(X)`

- *Description:* `X` is any term.
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

int/1: REGTYPE

The type of integers. The range of integers is `[-2^2147483616, 2^2147483616)`. Thus for all practical purposes, the range of integers can be considered infinite.

General properties: `int(T)`

– *The following properties hold globally:*

`int(T)` is side-effect `free`. (basic_props:sideff/2)

`int(T)`

– *If the following properties hold at call time:*

`T` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`int(T)` is evaluable at compile-time. (basic_props:eval/1)

All calls of the form `int(T)` are deterministic. (native_props:is_det/1)

`int(T)`

– *The following properties hold upon exit:*

`T` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `int(T)`

– *Description:* `T` is an integer.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

nnegint/1: REGTYPE

The type of non-negative integers, i.e., natural numbers.

General properties: `nnegint(T)`

– *The following properties hold globally:*

`nnegint(T)` is side-effect `free`. (basic_props:sideff/2)

`nnegint(T)`

– *If the following properties hold at call time:*

`T` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`nnegint(T)` is evaluable at compile-time. (basic_props:eval/1)

`nnegint(T)`

– *The following properties hold upon exit:*

`T` is a non-negative integer. (basic_props:nnegint/1)

– *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `nnegint(T)`

– *Description:* `T` is a non-negative integer.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

flt/1: REGTYPE

The type of floating-point numbers. The range of floats is the one provided by the C `double` type, typically `[4.9e-324, 1.8e+308]` (plus or minus). There are also three special values: Infinity, either positive or negative, represented as `1.0e1000` and `-1.0e1000`; and Not-a-number, which arises as the result of indeterminate operations, represented as `0.Nan`

General properties: `flt(T)`

– *The following properties hold globally:*

`flt(T)` is side-effect `free`. (basic_props:sideff/2)

`flt(T)`

– *If the following properties hold at call time:*

`T` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`flt(T)` is evaluable at compile-time. (basic_props:eval/1)

All calls of the form `flt(T)` are deterministic. (native_props:is_det/1)

`flt(T)`

– *The following properties hold upon exit:*

`T` is a float. (basic_props:flt/1)

– *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `flt(T)`

– *Description:* `T` is a float.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

num/1: REGTYPE

The type of numbers, that is, integer or floating-point.

General properties: `num(T)`

– *The following properties hold globally:*

`num(T)` is side-effect `free`. (basic_props:sideff/2)

`num(T)` is binding insensitive. (basic_props:bind_ins/1)

`num(T)`

– *If the following properties hold at call time:*

`T` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`num(T)` is evaluable at compile-time. (basic_props:eval/1)

All calls of the form `num(T)` are deterministic. (native_props:is_det/1)

`num(T)`

– *The following properties hold upon exit:*

`T` is a number. (basic_props:num/1)

– *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `num(T)`

– *Description:* `T` is a number.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

atm/1: REGTYPE

The type of atoms, or non-numeric constants. The size of atoms is unbound.

General properties: `atm(T)`

– *The following properties hold globally:*

`atm(T)` is side-effect `free`. (basic_props:sideff/2)

`atm(T)`

– *If the following properties hold at call time:*

`T` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`atm(T)` is evaluable at compile-time. (basic_props:eval/1)

All calls of the form `atm(T)` are deterministic. (native_props:is_det/1)

`atm(T)`

– *The following properties hold upon exit:*

`T` is an atom. (basic_props:atm/1)

– *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `atm(T)`

– *Description:* `T` is an atom.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

struct/1: REGTYPE

The type of compound terms, or terms with non-zeroary functors. By now there is a limit of 255 arguments.

General properties: `struct(T)`

– *The following properties hold globally:*

`struct(T)` is side-effect `free`. (basic_props:sideff/2)

`struct(T)`

– *If the following properties hold at call time:*

`T` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`struct(T)` is evaluable at compile-time. (basic_props:eval/1)

`struct(T)`

– *The following properties hold upon exit:*

`T` is a compound term. (basic_props:struct/1)

Usage: `struct(T)`

– *Description:* `T` is a compound term.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

gnd/1: REGTYPE

The type of all terms without variables.

General properties: `gnd(T)`

– *The following properties hold globally:*

`gnd(T)` is side-effect `free`. (basic_props:sideff/2)

`gnd(T)`

– *If the following properties hold at call time:*

`T` is currently ground (it contains no variables). (term_typing:ground/1)

then the following properties hold globally:

`gnd(T)` is evaluable at compile-time. (basic_props:eval/1)

All calls of the form `gnd(T)` are deterministic. (native_props:is_det/1)

`gnd(T)`

– *The following properties hold upon exit:*

`T` is ground. (basic_props:gnd/1)

– *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `gnd(T)`

– *Description:* `T` is ground.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

gndstr/1: REGTYPE

General properties: `gndstr(T)`

– *The following properties hold globally:*

`gndstr(T)` is side-effect `free`. (basic_props:sideff/2)

`gndstr(T)`

– *If the following properties hold at call time:*

`T` is currently ground (it contains no variables). (term_typing:ground/1)

then the following properties hold globally:

`gndstr(T)` is evaluable at compile-time. (basic_props:eval/1)

All calls of the form `gndstr(T)` are deterministic. (native_props:is_det/1)

`gndstr(T)`

– *The following properties hold upon exit:*

`T` is a ground compound term. (basic_props:gndstr/1)

Usage: `gndstr(T)`

– *Description:* `T` is a ground compound term.
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

constant/1: REGTYPE

General properties: `constant(T)`

– *The following properties hold globally:*
`constant(T)` is side-effect `free`. (basic_props:sideff/2)

`constant(T)`

– *If the following properties hold at call time:*
`T` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`constant(T)` is evaluable at compile-time. (basic_props:eval/1)
All calls of the form `constant(T)` are deterministic. (native_props:is_det/1)

`constant(T)`

– *The following properties hold upon exit:*
`T` is an atomic term (an atom or a number). (basic_props:constant/1)

Usage: `constant(T)`

– *Description:* `T` is an atomic term (an atom or a number).

callable/1: REGTYPE

General properties: `callable(T)`

– *The following properties hold globally:*
`callable(T)` is side-effect `free`. (basic_props:sideff/2)

`callable(T)`

– *If the following properties hold at call time:*
`T` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`callable(T)` is evaluable at compile-time. (basic_props:eval/1)
All calls of the form `callable(T)` are deterministic. (native_props:is_det/1)

`callable(T)`

– *The following properties hold upon exit:*
`T` is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage: `callable(T)`

– *Description:* `T` is a term which represents a goal, i.e., an atom or a structure.

operator_specifier/1: REGTYPE

The type and associativity of an operator is described by the following mnemonic atoms:

`xfx` Infix, non-associative: it is a requirement that both of the two subexpressions which are the arguments of the operator must be of *lower* precedence than the operator itself.

`xfy` Infix, right-associative: only the first (left-hand) subexpression must be of lower precedence; the right-hand subexpression can be of the *same* precedence as the main operator.

`yfx` Infix, left-associative: same as above, but the other way around.

`fx` Prefix, non-associative: the subexpression must be of *lower* precedence than the operator.

`fy` Prefix, associative: the subexpression can be of the *same* precedence as the operator.

`xf` Postfix, non-associative: the subexpression must be of *lower* precedence than the operator.

`yf` Postfix, associative: the subexpression can be of the *same* precedence as the operator.

General properties: `operator_specifier(X)`

– *The following properties hold globally:*

`operator_specifier(X)` is side-effect `free`. (basic_props:sideff/2)

`operator_specifier(X)`

– *If the following properties hold at call time:*

`X` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`operator_specifier(X)` is evaluable at compile-time. (basic_props:eval/1)

All calls of the form `operator_specifier(X)` are deterministic. (native_props:is_det/1)

Goal `operator_specifier(X)` produces `7` solutions. (native_props:relations/2)

`operator_specifier(T)`

– *The following properties hold upon exit:*

`T` specifies the type and associativity of an operator. (basic_props:operator_specifier/1)

Usage: `operator_specifier(X)`

– *Description:* `X` specifies the type and associativity of an operator.

list/1: REGTYPE

A list is formed with successive applications of the functor `'.'/2`, and its end is the atom `[]`. Defined as

```
list([]).
list([_1|L]) :-
        list(L).
```

General properties: `list(L)`

– *The following properties hold globally:*

`list(L)` is side-effect `free`. (basic_props:sideff/2)

`list(L)`

– *If the following properties hold at call time:*

`L` is currently ground (it contains no variables). (term_typing:ground/1)

then the following properties hold globally:

`list(L)` is evaluable at compile-time. (basic_props:eval/1)

All calls of the form `list(L)` are deterministic. (native_props:is_det/1)

`list(T)`

– *The following properties hold upon exit:*
`T` is a list. (basic_props:list/1)

Usage: `list(L)`

– *Description:* `L` is a list.

list/2: REGTYPE

`list(L,T)`

`L` is a list, and for all its elements, `T` holds.

Meta-predicate with arguments: `list(?,(pred 1))`.

General properties: `list(L,T)`

– *The following properties hold globally:*
`list(L,T)` is side-effect `free`. (basic_props:sideff/2)

`list(L,T)`

– *If the following properties hold at call time:*
`L` is currently ground (it contains no variables). (term_typing:ground/1)
`T` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`list(L,T)` is evaluable at compile-time. (basic_props:eval/1)

`list(X,T)`

– *The following properties hold upon exit:*
`X` is a list. (basic_props:list/1)

Usage: `list(L,T)`

– *Description:* `L` is a list of `T`s.

nlist/2: REGTYPE

Meta-predicate with arguments: `nlist(?,(pred 1))`.

General properties: `nlist(L,T)`

– *The following properties hold globally:*
`nlist(L,T)` is side-effect `free`. (basic_props:sideff/2)

`nlist(L,T)`

– *If the following properties hold at call time:*
`L` is currently ground (it contains no variables). (term_typing:ground/1)
`T` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`nlist(L,T)` is evaluable at compile-time. (basic_props:eval/1)

`nlist(X,T)`

– *The following properties hold upon exit:*
`X` is any term. (basic_props:term/1)

Usage: `nlist(L,T)`

– *Description:* `L` is `T` or a nested list of `T`s. Note that if `T` is term, this type is equivalent to term, this fact explain why we do not have a `nlist/1` type

member/2: PROPERTY

General properties: `member(X,L)`

– *The following properties hold globally:*

`member(X,L)` is side-effect `free`. (basic_props:sideff/2)

`member(X,L)` is binding insensitive. (basic_props:bind_ins/1)

`member(X,L)`

– *If the following properties hold at call time:*

`L` is a list. (basic_props:list/1)

then the following properties hold globally:

`member(X,L)` is evaluable at compile-time. (basic_props:eval/1)

`member(_X,L)`

– *The following properties hold upon exit:*

`L` is a list. (basic_props:list/1)

`member(X,L)`

– *If the following properties hold at call time:*

`L` is currently ground (it contains no variables). (term_typing:ground/1)

then the following properties hold upon exit:

`X` is currently ground (it contains no variables). (term_typing:ground/1)

Usage: `member(X,L)`

– *Description:* `X` is an element of `L`.

sequence/2: REGTYPE

A sequence is formed with zero, one or more occurrences of the operator `','/2`. For example, `a, b, c` is a sequence of three atoms, `a` is a sequence of one atom.

Meta-predicate with arguments: `sequence(?,(pred 1))`.

General properties: `sequence(S,T)`

– *The following properties hold globally:*

`sequence(S,T)` is side-effect `free`. (basic_props:sideff/2)

`sequence(S,T)`

– *If the following properties hold at call time:*

`S` is currently ground (it contains no variables). (term_typing:ground/1)

`T` is currently ground (it contains no variables). (term_typing:ground/1)

then the following properties hold globally:

`sequence(S,T)` is evaluable at compile-time. (basic_props:eval/1)

`sequence(E,T)`

– *The following properties hold upon exit:*

`E` is currently a term which is not a free variable. (term_typing:nonvar/1)

`T` is currently ground (it contains no variables). (term_typing:ground/1)

Usage: `sequence(S,T)`

– *Description:* `S` is a sequence of `T`s.

sequence_or_list/2: REGTYPE

Meta-predicate with arguments: `sequence_or_list(?,(pred 1))`.

General properties: `sequence_or_list(S,T)`

– *The following properties hold globally:*
`sequence_or_list(S,T)` is side-effect `free`. (basic_props:sideff/2)

`sequence_or_list(S,T)`

– *If the following properties hold at call time:*
`S` is currently ground (it contains no variables). (term_typing:ground/1)
`T` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`sequence_or_list(S,T)` is evaluable at compile-time. (basic_props:eval/1)

`sequence_or_list(E,T)`

– *The following properties hold upon exit:*
`E` is currently a term which is not a free variable. (term_typing:nonvar/1)
`T` is currently ground (it contains no variables). (term_typing:ground/1)

Usage: `sequence_or_list(S,T)`

– *Description:* `S` is a sequence or list of `T`s.

character_code/1: REGTYPE

General properties: `character_code(T)`

– *The following properties hold globally:*
`character_code(T)` is side-effect `free`. (basic_props:sideff/2)

`character_code(T)`

– *If the following properties hold at call time:*
`T` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`character_code(T)` is evaluable at compile-time. (basic_props:eval/1)

`character_code(I)`

– *The following properties hold upon exit:*
`I` is an integer which is a character code. (basic_props:character_code/1)

Usage: `character_code(T)`

– *Description:* `T` is an integer which is a character code.

string/1: REGTYPE

A string is a list of character codes. The usual syntax for strings `"string"` is allowed, which is equivalent to `[0's,0't,0'r,0'i,0'n,0'g]` or `[115,116,114,105,110,103]`. There is also a special Ciao syntax when the list is not complete: `"st"||R` is equivalent to `[0's,0't|R]`.

General properties: `string(T)`

– *The following properties hold globally:*
`string(T)` is side-effect `free`. (basic_props:sideff/2)

`string(T)`

– *If the following properties hold at call time:*
`T` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`string(T)` is evaluable at compile-time. (basic_props:eval/1)

`string(T)`

– *The following properties hold upon exit:*
`T` is a string (a list of character codes). (basic_props:string/1)

Usage: `string(T)`

– *Description:* `T` is a string (a list of character codes).

num_code/1: REGTYPE

These are the ASCII codes which can appear in decimal representation of floating point and integer numbers, including scientific notation and fractionary part.

predname/1: REGTYPE

General properties: `predname(P)`

– *The following properties hold globally:*
`predname(P)` is side-effect `free`. (basic_props:sideff/2)

`predname(P)`

– *If the following properties hold at call time:*
`P` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`predname(P)` is evaluable at compile-time. (basic_props:eval/1)

`predname(P)`

– *The following properties hold upon exit:*
`P` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

Usage: `predname(P)`

– *Description:* `P` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

atm_or_atm_list/1: REGTYPE

General properties: `atm_or_atm_list(T)`

– *The following properties hold globally:*
`atm_or_atm_list(T)` is side-effect `free`. (basic_props:sideff/2)

`atm_or_atm_list(T)`

– *If the following properties hold at call time:*
`T` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`atm_or_atm_list(T)` is evaluable at compile-time. (basic_props:eval/1)

`atm_or_atm_list(T)`

– *The following properties hold upon exit:*
`T` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

Usage: `atm_or_atm_list(T)`

– *Description:* `T` is an atom or a list of atoms.

compat/2: PROPERTY

This property captures the notion of type or property compatibility. The instantiation or constraint state of the term is compatible with the given property, in the sense that assuming that imposing that property on the term does not render the store inconsistent. For example, terms `X` (i.e., a free variable), `[Y|Z]`, and `[Y,Z]` are all compatible with the regular type `list/1`, whereas the terms `f(a)` and `[1|2]` are not.

Meta-predicate with arguments: `compat(?,(pred 1))`.

General properties: `compat(Term,Prop)`

– *If the following properties hold at call time:*
`Term` is currently ground (it contains no variables). (term_typing:ground/1)
`Prop` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`compat(Term,Prop)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `compat(Term,Prop)`

– *Description:* `Term` is *compatible* with `Prop`

inst/2: PROPERTY

Meta-predicate with arguments: `inst(?,(pred 1))`.

General properties: `inst(Term,Prop)`

– *The following properties hold globally:*
`inst(Term,Prop)` is side-effect `free`. (basic_props:sideff/2)

`inst(Term,Prop)`

– *If the following properties hold at call time:*
`Term` is currently ground (it contains no variables). (term_typing:ground/1)
`Prop` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`inst(Term,Prop)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `inst(Term,Prop)`

– *Description:* `Term` is instantiated enough to satisfy `Prop`.

iso/1: PROPERTY

Meta-predicate with arguments: `iso(goal)`.

General properties: `iso(G)`

– *The following properties hold globally:*
`iso(G)` is side-effect `free`. (basic_props:sideff/2)

Usage: `iso(G)`

– *Description: Complies with the ISO-Prolog standard.*

deprecated/1: PROPERTY

Specifies that the predicate marked with this global property has been deprecated, i.e., its use is not recommended any more since it will be deleted at a future date. Typically this is done because its functionality has been superseded by another predicate.

Meta-predicate with arguments: `deprecated(goal)`.

General properties: `deprecated(G)`

– *The following properties hold globally:*
`deprecated(G)` is side-effect `free`. (basic_props:sideff/2)

Usage: `deprecated(G)`

– *Description:* **DEPRECATED.**

not_further_inst/2: PROPERTY

Meta-predicate with arguments: `not_further_inst(goal,?)`.

General properties: `not_further_inst(G,V)`

– *The following properties hold globally:*
`not_further_inst(G,V)` is side-effect `free`. (basic_props:sideff/2)

Usage: `not_further_inst(G,V)`

– *Description:* `V` is not further instantiated.

sideff/2: PROPERTY

`sideff(G,X)`

Declares that `G` is side-effect free (if its execution has no observable result other than its success, its failure, or its abortion), soft (if its execution may have other observable results which, however, do not affect subsequent execution, e.g., input/output), or hard (e.g., assert/retract).

Meta-predicate with arguments: `sideff(goal,?)`.

General properties: `sideff(G,X)`

– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)
`sideff(G,X)` is side-effect `free`. (basic_props:sideff/2)

Usage: `sideff(G,X)`

– *Description:* `G` is side-effect `X`.

– *If the following properties hold at call time:*
`X` is an element of `[free,soft,hard]`. (basic_props:member/2)

regtype/1: PROPERTY

Meta-predicate with arguments: `regtype goal`.

General properties: `regtype G`

– *The following properties hold globally:*
`regtype G` is side-effect `free`. (basic_props:sideff/2)

Usage: `regtype G`

– *Description:* Defines a regular type.

native/1: PROPERTY

Meta-predicate with arguments: `native(goal)`.

General properties: `native(P)`

– *The following properties hold globally:*
`native(P)` is side-effect `free`. (basic_props:sideff/2)

Usage: `native(Pred)`

– *Description:* This predicate is understood natively by CiaoPP.

native/2: PROPERTY

Meta-predicate with arguments: `native(goal,?)`.

General properties: `native(P,K)`

– *The following properties hold globally:*
`native(P,K)` is side-effect `free`. (basic_props:sideff/2)

Usage: `native(Pred,Key)`

– *Description:* This predicate is understood natively by CiaoPP as `Key`.

no_rtcheck/1: PROPERTY

Meta-predicate with arguments: `no_rtcheck(goal)`.

General properties: `no_rtcheck(G)`

– *The following properties hold globally:*
`no_rtcheck(G)` is side-effect `free`. (basic_props:sideff/2)

Usage: `no_rtcheck(G)`

– *Description:* Declares that the assertion in which this comp property appears must not be checked at run-time.

eval/1: PROPERTY

Meta-predicate with arguments: `eval(goal)`.

Usage: `eval(Goal)`

– *Description:* `Goal` is evaluable at compile-time.

equiv/2: PROPERTY

Meta-predicate with arguments: `equiv(goal,goal)`.

Usage: `equiv(Goal1,Goal2)`

– *Description:* `Goal1` is equivalent to `Goal2`.

bind_ins/1: PROPERTY

Meta-predicate with arguments: `bind_ins(goal)`.

Usage: `bind_ins(Goal)`

– *Description:* `Goal` is binding insensitive.

error_free/1: PROPERTY

Meta-predicate with arguments: `error_free(goal)`.

Usage: `error_free(Goal)`

– *Description:* `Goal` is error free.

memo/1: PROPERTY

Meta-predicate with arguments: `memo(goal)`.

Usage: `memo(Goal)`

– *Description:* `Goal` should be memoized (not unfolded).

filter/2: PROPERTY

Usage: `filter(Vars,Goal)`

– *Description:* `Vars` should be filtered during global control).

flag_values/1: REGTYPE

Usage: `flag_values(X)`

– *Description:* Define the valid flag values

pe_type/1: PROPERTY

Meta-predicate with arguments: `pe_type(goal)`.

Usage: `pe_type(Goal)`

– *Description:* `Goal` will be filtered in partial evaluation time according to the PE types defined in the assertion.

15.3 Known bugs and planned improvements (`basic_props`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

16 Extra-logical properties for typing

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This library contains traditional Prolog predicates for testing types. They depend on the state of instantiation of their arguments, thus being of extra-logical nature.

16.1 Usage and interface (`term_typing`)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Properties:*
    `var/1`, `nonvar/1`, `atom/1`, `integer/1`, `float/1`, `number/1`, `atomic/1`, `ground/1`, `type/2`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

16.2 Documentation on exports (`term_typing`)

var/1: PROPERTY

General properties:

- *The following properties hold globally:*
  All calls of the form `var(Arg1)` are deterministic. (native_props:is_det/1)
  Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

`var(X)`

- *The following properties hold globally:*
  `X` is not further instantiated. (basic_props:not_further_inst/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)
  `var(X)` is side-effect `free`. (basic_props:sideff/2)

`var(X)`

- *If the following properties hold at call time:*
  `X` is currently a term which is not a free variable. (term_typing:nonvar/1)
  then the following properties hold globally:
  `var(X)` is evaluable at compile-time. (basic_props:eval/1)

`var(X)`

- *If the following properties hold at call time:*
  `X` is currently a term which is not a free variable. (term_typing:nonvar/1)
  then the following properties hold globally:
  `var(X)` is equivalent to `fail`. (basic_props:equiv/2)

`var(X)`

– *If the following properties hold at call time:*
`X` is a free variable. (term_typing:var/1)
then the following properties hold globally:
`var(X)` is equivalent to `true`. (basic_props:equiv/2)

Usage: `var(X)`
– *Description:* `X` is a free variable.
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP as `free(X)`. (basic_props:native/2)

nonvar/1: PROPERTY

General properties:
– *The following properties hold globally:*
All calls of the form `nonvar(Arg1)` are deterministic. (native_props:is_det/1)

`nonvar(X)`
– *The following properties hold globally:*
`X` is not further instantiated. (basic_props:not_further_inst/2)
`nonvar(X)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

`nonvar(X)`
– *If the following properties hold at call time:*
`X` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`nonvar(X)` is evaluable at compile-time. (basic_props:eval/1)

`nonvar(T)`
– *If the following properties hold at call time:*
`T` is a free variable. (term_typing:var/1)
then the following properties hold globally:
`nonvar(T)` is equivalent to `fail`. (basic_props:equiv/2)

`nonvar(T)`
– *If the following properties hold at call time:*
`T` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`nonvar(T)` is equivalent to `true`. (basic_props:equiv/2)

Usage: `nonvar(X)`
– *Description:* `X` is currently a term which is not a free variable.
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP as `not_free(X)`. (basic_props:native/2)

atom/1: PROPERTY

General properties: `atom(X)`

– *The following properties hold upon exit:*

`X` is an atom. (basic_props:atm/1)

– *The following properties hold globally:*

All calls of the form `atom(Arg1)` are deterministic. (native_props:is_det/1)

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

`atom(X)`

– *The following properties hold globally:*

`X` is not further instantiated. (basic_props:not_further_inst/2)

`atom(X)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`atom(X)`

– *If the following properties hold at call time:*

`X` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`atom(X)` is evaluable at compile-time. (basic_props:eval/1)

`atom(T)`

– *If the following properties hold at call time:*

`T` is a free variable. (term_typing:var/1)

then the following properties hold globally:

`atom(T)` is equivalent to `fail`. (basic_props:equiv/2)

Usage: `atom(X)`

– *Description:* `X` is currently instantiated to an atom.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

integer/1: PROPERTY

General properties: `integer(X)`

– *The following properties hold upon exit:*

`X` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

All calls of the form `integer(Arg1)` are deterministic. (native_props:is_det/1)

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

`integer(X)`

– *The following properties hold globally:*

`X` is not further instantiated. (basic_props:not_further_inst/2)

`integer(X)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`integer(X)`

– *If the following properties hold at call time:*
`X` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`integer(X)` is evaluable at compile-time. (basic_props:eval/1)

`integer(T)`
– *If the following properties hold at call time:*
`T` is a free variable. (term_typing:var/1)
then the following properties hold globally:
`integer(T)` is equivalent to `fail`. (basic_props:equiv/2)

Usage: `integer(X)`
– *Description:* `X` is currently instantiated to an integer.
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

float/1: PROPERTY

General properties: `float(X)`
– *The following properties hold upon exit:*
`X` is a float. (basic_props:flt/1)
– *The following properties hold globally:*
All calls of the form `float(Arg1)` are deterministic. (native_props:is_det/1)
Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

`float(X)`
– *The following properties hold globally:*
`X` is not further instantiated. (basic_props:not_further_inst/2)
`float(X)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

`float(X)`
– *If the following properties hold at call time:*
`X` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`float(X)` is evaluable at compile-time. (basic_props:eval/1)

`float(T)`
– *If the following properties hold at call time:*
`T` is a free variable. (term_typing:var/1)
then the following properties hold globally:
`float(T)` is equivalent to `fail`. (basic_props:equiv/2)

Usage: `float(X)`
– *Description:* `X` is currently instantiated to a float.
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

number/1: PROPERTY

General properties: `number(X)`

– *The following properties hold upon exit:*

`X` is a number. (basic_props:num/1)

– *The following properties hold globally:*

All calls of the form `number(Arg1)` are deterministic. (native_props:is_det/1)

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

`number(X)`

– *The following properties hold globally:*

`X` is not further instantiated. (basic_props:not_further_inst/2)

`number(X)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`number(X)`

– *If the following properties hold at call time:*

`X` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`number(X)` is evaluable at compile-time. (basic_props:eval/1)

`number(T)`

– *If the following properties hold at call time:*

`T` is a free variable. (term_typing:var/1)

then the following properties hold globally:

`number(T)` is equivalent to `fail`. (basic_props:equiv/2)

Usage: `number(X)`

– *Description:* `X` is currently instantiated to a number.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

atomic/1: PROPERTY

General properties: `atomic(T)`

– *The following properties hold upon exit:*

`T` is an atomic term (an atom or a number). (basic_props:constant/1)

– *The following properties hold globally:*

All calls of the form `atomic(Arg1)` are deterministic. (native_props:is_det/1)

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

`atomic(X)`

– *The following properties hold globally:*

`X` is not further instantiated. (basic_props:not_further_inst/2)

`atomic(X)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`atomic(X)`

– *If the following properties hold at call time:*
`X` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`atomic(X)` is evaluable at compile-time. (basic_props:eval/1)

`atomic(T)`
– *If the following properties hold at call time:*
`T` is a free variable. (term_typing:var/1)
then the following properties hold globally:
`atomic(T)` is equivalent to `fail`. (basic_props:equiv/2)

Usage: `atomic(X)`
– *Description:* `X` is currently instantiated to an atom or a number.
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

ground/1: PROPERTY

General properties: `ground(X)`
– *The following properties hold upon exit:*
`X` is ground. (basic_props:gnd/1)
– *The following properties hold globally:*
All calls of the form `ground(Arg1)` are deterministic. (native_props:is_det/1)
Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

`ground(X)`
– *The following properties hold globally:*
`X` is not further instantiated. (basic_props:not_further_inst/2)
`ground(X)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

`ground(X)`
– *If the following properties hold at call time:*
`X` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`ground(X)` is evaluable at compile-time. (basic_props:eval/1)

`ground(X)`
– *If the following properties hold at call time:*
`X` is a free variable. (term_typing:var/1)
then the following properties hold globally:
`ground(X)` is equivalent to `fail`. (basic_props:equiv/2)

`ground(X)`
– *If the following properties hold at call time:*
`X` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`ground(X)` is equivalent to `true`. (basic_props:equiv/2)

Usage: `ground(X)`

- *Description:* `X` is currently ground (it contains no variables).
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

type/2: PROPERTY

General properties: `type(X,Y)`

- *The following properties hold upon exit:*
  `Y` is an atom. (basic_props:atm/1)
- *The following properties hold globally:*
  All calls of the form `type(Arg1,Arg2)` are deterministic. (native_props:is_det/1)
- *The following properties hold globally:*
  `type(Arg1,Arg2)` is side-effect `free`. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

`type(X,Y)`

- *If the following properties hold at call time:*
  `X` is currently a term which is not a free variable. (term_typing:nonvar/1)
  then the following properties hold globally:
  `type(X,Y)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `type(X,Y)`

- *Description:* `X` is internally of type `Y` (`var`, `attv`, `float`, `integer`, `structure`, `atom` or `list`).
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

16.3 Known bugs and planned improvements (`term_typing`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

17 Basic term manipulation

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This module provides basic term manipulation.

17.1 Usage and interface (term_basic)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    \=/2, arg/3, functor/3, =../2, copy_term/2, copy_term_nat/2, C/3.
  - *Properties:*
    =/2, const_head/1.
  - *Regular Types:*
    non_empty_list/1, list_functor/1.
- **Other modules used:**
  - *System library modules:*
    assertions/native_props.

17.2 Documentation on exports (term_basic)

=/2: PROPERTY

〈• ISO •〉 A property, defined as follows:

```
X=Y :-
        X=Y.
```

General properties: X=Y

- *Description:* X and Y unify.
- *The following properties hold globally:*

X=Y is side-effect free.	(basic_props:sideff/2)
This predicate is understood natively by CiaoPP.	(basic_props:native/1)
X=Y is evaluable at compile-time.	(basic_props:eval/1)
All calls of the form X=Y are deterministic.	(native_props:is_det/1)
Goal X=Y produces inf solutions.	(native_props:relations/2)
Indicates the type of test that a predicate performs. Required by the nonfailure analyisis.	(native_props:test_type/2)

\=/2: PREDICATE

General properties: X\=Y

– *If the following properties hold at call time:*
`X` is currently ground (it contains no variables). (term_typing:ground/1)
`Y` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`X\=Y` is evaluable at compile-time. (basic_props:eval/1)

`X\=Y`

– *The following properties hold globally:*
`X\=Y` is side-effect `free`. (basic_props:sideff/2)

Usage: `X\=Y` ⟨• ISO •⟩

– *Description:* `X` and `Y` are not unifiable.
– *The following properties hold globally:*
`X\=Y` is side-effect `free`. (basic_props:sideff/2)
`X\=Y` is binding insensitive. (basic_props:bind_ins/1)
All calls of the form `X\=Y` are deterministic. (native_props:is_det/1)

arg/3: PREDICATE

Usage 1: `arg(ArgNo,Term,Arg)`

– *The following properties should hold at call time:*
`ArgNo` is a number. (basic_props:num/1)
– *The following properties hold globally:*
All calls of the form `arg(ArgNo,Term,Arg)` are deterministic. (native_props:is_det/1)
Goal `arg(ArgNo,Term,Arg)` produces `inf` solutions. (native_props:relations/2)

Usage 2: `arg(ArgNo,Term,Arg)`

– *The following properties should hold at call time:*
`ArgNo` is a number. (basic_props:num/1)
`Term` is ground. (basic_props:gnd/1)
– *The following properties hold upon exit:*
`Arg` is ground. (basic_props:gnd/1)

Usage 3: `arg(ArgNo,Term,Arg)` ⟨• ISO •⟩

– *Description:* Argument `ArgNo` of the term `Term` is `Arg`.
– *The following properties should hold at call time:*
`ArgNo` is a non-negative integer. (basic_props:nnegint/1)
`Term` is a compound term. (basic_props:struct/1)
– *The following properties hold globally:*
`arg(ArgNo,Term,Arg)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)
`arg(ArgNo,Term,Arg)` is binding insensitive. (basic_props:bind_ins/1)

Usage 4: `arg(ArgNo,Term,Arg)`

– *The following properties should hold at call time:*
`ArgNo` is a non-negative integer. (basic_props:nnegint/1)
`Term` is ground. (basic_props:gnd/1)
– *The following properties hold upon exit:*
`Arg` is ground. (basic_props:gnd/1)

functor/3: PREDICATE

⟨• ISO •⟩

General properties:

– *The following properties hold globally:*

`functor(Arg1,Arg2,Arg3)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

All calls of the form `functor(Arg1,Arg2,Arg3)` are deterministic. (native_props:is_det/1)

Usage 1: `functor(Term,Name,Arity)` ⟨• ISO •⟩

– *The following properties should hold at call time:*

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

– *The following properties hold upon exit:*

`Name` is an atom. (basic_props:atm/1)

`Arity` is a non-negative integer. (basic_props:nnegint/1)

– *The following properties hold globally:*

`Term` is not further instantiated. (basic_props:not_further_inst/2)

`functor(Term,Name,Arity)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`functor(Term,Name,Arity)` is binding insensitive. (basic_props:bind_ins/1)

`functor(Term,Name,Arity)` is evaluable at compile-time. (basic_props:eval/1)

Usage 2: `functor(Term,Name,Arity)` ⟨• ISO •⟩

– *Description:* The principal functor of the term `Term` has name `Name` and arity `Arity`.

– *The following properties should hold at call time:*

`Name` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Arity` is currently a term which is not a free variable. (term_typing:nonvar/1)

– *The following properties hold upon exit:*

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Name` is an atom. (basic_props:atm/1)

`Arity` is a non-negative integer. (basic_props:nnegint/1)

– *The following properties hold globally:*

`functor(Term,Name,Arity)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`functor(Term,Name,Arity)` is binding insensitive. (basic_props:bind_ins/1)

`functor(Term,Name,Arity)` is evaluable at compile-time. (basic_props:eval/1)

All the calls of the form `functor(Term,Name,Arity)` do not fail. (native_props:not_fails/1)

Usage 3: `functor(Term,Name,Arity)`

– *The following properties should hold at call time:*

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Name` is a free variable. (term_typing:var/1)

`Arity` is a free variable. (term_typing:var/1)

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

– *The following properties hold upon exit:*

`1` is the size of argument `Name`, for any approximation. (native_props:size/2)

`arity(Term)` is the size of argument `Arity`, for any approximation. (native_props:size/2)

– *The following properties hold globally:*

`arity` is the metric of the variable `Term`, for any approximation. (native_props:size_metric/3)

=../2: PREDICATE

General properties:

– *If the following properties hold at call time:*

`Arg1` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`Arg1=..Arg2` is evaluable at compile-time. (basic_props:eval/1)

`Arg1=..List`

– *If the following properties hold at call time:*

`List` is a list. (basic_props:list/1)

term_basic:const_head(List) (term_basic:const_head/1)

then the following properties hold globally:

`Arg1=..List` is evaluable at compile-time. (basic_props:eval/1)

Usage: `Term=..List` ⟨• ISO •⟩

– *Description:* The functor and arguments of the term `Term` comprise the list `List`.

– *The following properties hold upon exit:*

`List` is a list. (basic_props:list/1)

– *The following properties hold globally:*

`Term=..List` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

non_empty_list/1: REGTYPE

Usage: `non_empty_list(A)`

– *Description:* A list that is not the empty list [].

copy_term/2: PREDICATE

General properties: `copy_term(Term,Copy)`

– *If the following properties hold at call time:*

`Term` is currently ground (it contains no variables). (term_typing:ground/1)

then the following properties hold globally:

`copy_term(Term,Copy)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `copy_term(Term,Copy)` ⟨• ISO •⟩

– *Description:* `Copy` is a renaming of `Term`, such that brand new variables have been substituted for all variables in `Term`. If any of the variables of `Term` have attributes, the copied variables will have copies of the attributes as well. It behaves as if defined by:

```
:- data 'copy of'/1.

copy_term(X, Y) :-
        asserta_fact('copy of'(X)),
        retract_fact('copy of'(Y)).
```

– *The following properties should hold globally:*

`copy_term(Term,Copy)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

copy_term_nat/2: PREDICATE

Usage: `copy_term_nat(Term,Copy)`

– *Description:* Same as `copy_term/2`, except that attributes of variables are not copied.

– *The following properties hold globally:*

`copy_term_nat(Term,Copy)` is side-effect `free`. (basic_props:sideff/2)

C/3: PREDICATE

General properties: `C(S1,Terminal,S2)`

– *If the following properties hold at call time:*

A list that is not the empty list []. (term_basic:non_empty_list/1)

then the following properties hold upon exit:

`S2` is a list. (basic_props:list/1)

– *If the following properties hold at call time:*

`Arg1` is currently a term which is not a free variable. (term_typing:nonvar/1)

then the following properties hold globally:

`C(Arg1,Arg2,Arg3)` is evaluable at compile-time. (basic_props:eval/1)

Usage 1: `C(S1,Terminal,S2)`

– *The following properties hold upon exit:*

term_basic:list_functor(S1) (term_basic:list_functor/1)

Usage 2: `C(S1,Terminal,S2)`

– *Description:* `S1` is connected by the terminal `Terminal` to `S2`. Internally used in *DCG grammar rules*. Defined as if by the single clause: `'C'([X|S], X, S).`

– *The following properties hold globally:*

`C(S1,Terminal,S2)` is side-effect `free`. (basic_props:sideff/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

const_head/1: PROPERTY

A property, defined as follows:

```
const_head([Head|_1]) :-
        constant(Head).
```

list_functor/1: REGTYPE

A regular type, defined as follows:

```
list_functor([A|B]) :-
        term(A),
        term(B).
```

17.3 Known bugs and planned improvements (term_basic)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

18 Comparing terms

Author(s): Daniel Cabeza, Manuel Hermenegildo.

These built-in predicates are extra-logical. They treat uninstantiated variables as objects with values which may be compared, and they never instantiate those variables. They should *not* be used when what you really want is arithmetic comparison or unification.

The predicates make reference to a *standard total ordering* of terms, which is as follows:

- Variables, by age (roughly, oldest first – the order is *not* related to the names of variables).
- Floats, in numeric order (e.g. -1.0 is put before 1.0).
- Integers, in numeric order (e.g. -1 is put before 1).
- Atoms, in alphabetical (i.e. character code) order.
- Compound terms, ordered first by arity, then by the name of the principal functor, then by the arguments in left-to-right order. Recall that lists are equivalent to compound terms with principal functor `'.'/2`.

For example, here is a list of terms in standard order:

```
[ X, -1.0, -9, 1, bar, foo, [1], X = Y, foo(0,2), bar(1,1,1) ]
```

18.1 Usage and interface (term_compare)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `\==/2`, `@</2`, `@=</2`, `@>/2`, `@>=/2`, `compare/3`.
  - *Properties:*
    `==/2`.
  - *Regular Types:*
    `comparator/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

18.2 Documentation on exports (term_compare)

==/2: PROPERTY

General properties: `Term1==Term2`

- *The following properties hold globally:*
  `Term1` is not further instantiated. (basic_props:not_further_inst/2)
  `Term2` is not further instantiated. (basic_props:not_further_inst/2)
  `Term1==Term2` is side-effect `free`. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

`Term1==Term2`

– *If the following properties hold at call time:*
`Term1` is currently ground (it contains no variables). (term_typing:ground/1)
`Term2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`Term1==Term2` is evaluable at compile-time. (basic_props:eval/1)

`Term1==Term2`

– *The following properties hold globally:*
All calls of the form `Term1==Term2` are deterministic. (native_props:is_det/1)
Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `Term1==Term2`

– *Description:* The terms `Term1` and `Term2` are strictly identical.

\==/2: PREDICATE

General properties: `Term1\==Term2`

– *If the following properties hold at call time:*
`Term1` is currently ground (it contains no variables). (term_typing:ground/1)
`Term2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`Term1\==Term2` is evaluable at compile-time. (basic_props:eval/1)

`Term1\==Term2`

– *The following properties hold globally:*
All calls of the form `Term1\==Term2` are deterministic. (native_props:is_det/1)
Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `Term1\==Term2`

– *Description:* The terms `Term1` and `Term2` are not strictly identical.
– *The following properties hold globally:*
`Term1` is not further instantiated. (basic_props:not_further_inst/2)
`Term2` is not further instantiated. (basic_props:not_further_inst/2)
`Term1\==Term2` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

@</2: PREDICATE

General properties: `Term1@<Term2`

– *If the following properties hold at call time:*
`Term1` is currently ground (it contains no variables). (term_typing:ground/1)
`Term2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`Term1@<Term2` is evaluable at compile-time. (basic_props:eval/1)

Usage: `Term1@<Term2`

– *Description:* The term `Term1` precedes the term `Term2` in the standard order.
– *The following properties hold globally:*
`Term1` is not further instantiated. (basic_props:not_further_inst/2)
`Term2` is not further instantiated. (basic_props:not_further_inst/2)
`Term1@<Term2` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

`@=</2:` PREDICATE

General properties: `Term1@=<Term2`

– *If the following properties hold at call time:*
`Term1` is currently ground (it contains no variables). (term_typing:ground/1)
`Term2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`Term1@=<Term2` is evaluable at compile-time. (basic_props:eval/1)

Usage: `Term1@=<Term2`

– *Description:* The term `Term1` precedes or is identical to the term `Term2` in the standard order.
– *The following properties hold globally:*
`Term1` is not further instantiated. (basic_props:not_further_inst/2)
`Term2` is not further instantiated. (basic_props:not_further_inst/2)
`Term1@=<Term2` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

`@>/2:` PREDICATE

General properties: `Term1@>Term2`

– *If the following properties hold at call time:*
`Term1` is currently ground (it contains no variables). (term_typing:ground/1)
`Term2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`Term1@>Term2` is evaluable at compile-time. (basic_props:eval/1)

Usage: `Term1@>Term2`

– *Description:* The term `Term1` follows the term `Term2` in the standard order.
– *The following properties hold globally:*
`Term1` is not further instantiated. (basic_props:not_further_inst/2)
`Term2` is not further instantiated. (basic_props:not_further_inst/2)
`Term1@>Term2` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

`@>=/2:` PREDICATE

General properties: `Term1@>=Term2`

– *If the following properties hold at call time:*
`Term1` is currently ground (it contains no variables). (term_typing:ground/1)
`Term2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`Term1@>=Term2` is evaluable at compile-time. (basic_props:eval/1)

Usage: `Term1@>=Term2`

– *Description:* The term `Term1` follows or is identical to the term `Term2` in the standard order.
– *The following properties hold globally:*
`Term1` is not further instantiated. (basic_props:not_further_inst/2)
`Term2` is not further instantiated. (basic_props:not_further_inst/2)
`Term1@>=Term2` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

compare/3: PREDICATE

`compare(Op,Term1,Term2)`

`Op` is the result of comparing the terms `Term1` and `Term2`.

General properties: `compare(Op,Term1,Term2)`

– *If the following properties hold at call time:*
`Term1` is currently ground (it contains no variables). (term_typing:ground/1)
`Term2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`compare(Op,Term1,Term2)` is evaluable at compile-time. (basic_props:eval/1)

Usage:

– *Calls should, and exit will be compatible with:*
`Op` is an atom. (basic_props:atm/1)
– *The following properties should hold at call time:*
`Term1` is any term. (basic_props:term/1)
`Term2` is any term. (basic_props:term/1)
– *The following properties hold upon exit:*
`Op` is an atom. (basic_props:atm/1)
`Term1` is any term. (basic_props:term/1)
`Term2` is any term. (basic_props:term/1)
term_compare:comparator(Op) (term_compare:comparator/1)
`Term1` is any term. (basic_props:term/1)
`Term2` is any term. (basic_props:term/1)
– *The following properties hold globally:*
`Term1` is not further instantiated. (basic_props:not_further_inst/2)
`Term2` is not further instantiated. (basic_props:not_further_inst/2)
`compare(Op,Term1,Term2)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

comparator/1: REGTYPE

A regular type, defined as follows:

```
comparator(=).
comparator(>).
comparator(<).
```

18.3 Known bugs and planned improvements (`term_compare`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

19 Basic predicates handling names of constants

Author(s): The CLIP Group.

The Ciao system provides builtin predicates which allow dealing with names of constants (atoms or numbers). Note that sometimes strings (character code lists) are more suitable to handle sequences of characters.

19.1 Usage and interface (atomic_basic)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `name/2`, `atom_codes/2`, `number_codes/2`, `number_codes/3`, `atom_number/2`, `atom_number/3`, `atom_length/2`, `atom_concat/3`, `sub_atom/4`.
  - *Properties:*
    `number_codes/3`.
  - *Regular Types:*
    `valid_base/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

19.2 Documentation on exports (atomic_basic)

name/2: PREDICATE

`name(Const,String)`

`String` is the list of the ASCII codes of the characters comprising the name of `Const`. Note that if `Const` is an atom whose name can be interpreted as a number (e.g. '96'), the predicate is not reversible, as that atom will not be constructed when `Const` is uninstantiated. Thus it is recommended that new programs use the ISO-compliant predicates `atom_codes/2` or `number_codes/2`, as these predicates do not have this inconsistency.

General properties:

- *The following properties hold globally:*
  `name(Const,String)` is side-effect `free`. (basic_props:sideff/2)
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 1:

- *Calls should, and exit will be compatible with:*
  `String` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*
  `Const` is an atomic term (an atom or a number). (basic_props:constant/1)
- *The following properties hold upon exit:*
  `String` is a string (a list of character codes). (basic_props:string/1)

– *The following properties hold globally:*
`name(Const,String)` is evaluable at compile-time. (basic_props:eval/1)

Usage 2:
– *Description:* If `String` can be interpreted as a number, `Const` is unified with that number, otherwise with the atom whose name is `String`.
– *The following properties should hold at call time:*
`Const` is a free variable. (term_typing:var/1)
`String` is a string (a list of character codes). (basic_props:string/1)
– *The following properties hold upon exit:*
`Const` is an atomic term (an atom or a number). (basic_props:constant/1)
– *The following properties hold globally:*
`name(Const,String)` is evaluable at compile-time. (basic_props:eval/1)

atom_codes/2: PREDICATE

`atom_codes(Atom,String)` ⟨• ISO •⟩

`String` is the list of the ASCII codes of the characters comprising the name of `Atom`.

General properties: `atom_codes(A,B)`
– *If the following properties should hold at call time:*
term_basic:A=ao (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B=[97,241,111] (term_basic:= /2)
then the following properties should hold globally:
All the calls of the form `atom_codes(A,B)` do not fail. (native_props:not_fails/1)
All calls of the form `atom_codes(A,B)` are deterministic. (native_props:is_det/1)
– *The following properties hold globally:*
`atom_codes(Atom,String)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)
All calls of the form `atom_codes(Atom,String)` are deterministic. (native_props:is_det/1)

Usage 1:
– *Calls should, and exit will be compatible with:*
`String` is a string (a list of character codes). (basic_props:string/1)
– *The following properties should hold at call time:*
`Atom` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`String` is a string (a list of character codes). (basic_props:string/1)
– *The following properties hold globally:*
`atom_codes(Atom,String)` is evaluable at compile-time. (basic_props:eval/1)

Usage 2:
– *Calls should, and exit will be compatible with:*
`Atom` is an atom. (basic_props:atm/1)
– *The following properties should hold at call time:*
`String` is a string (a list of character codes). (basic_props:string/1)

– *The following properties hold upon exit:*
`Atom` is an atom. (basic_props:atm/1)
– *The following properties hold globally:*
`atom_codes(Atom,String)` is evaluable at compile-time. (basic_props:eval/1)

number_codes/2: PREDICATE

`number_codes(Number,String)` ⟨• ISO •⟩

`String` is the list of the ASCII codes of the characters comprising a representation of `Number`.

General properties:

– *The following properties hold globally:*
`number_codes(Number,String)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

`number_codes(A,B)`

– *If the following properties hold at call time:*
`A` is an integer. (basic_props:int/1)
then the following properties hold upon exit:
`B` is a list of `num_code`s. (basic_props:list/2)

Usage 1:

– *Calls should, and exit will be compatible with:*
`String` is a string (a list of character codes). (basic_props:string/1)
– *The following properties should hold at call time:*
`Number` is a number. (basic_props:num/1)
– *The following properties hold upon exit:*
`String` is a string (a list of character codes). (basic_props:string/1)
– *The following properties hold globally:*
`number_codes(Number,String)` is evaluable at compile-time. (basic_props:eval/1)

Usage 2:

– *The following properties should hold at call time:*
`Number` is a free variable. (term_typing:var/1)
`String` is a string (a list of character codes). (basic_props:string/1)
– *The following properties hold upon exit:*
`Number` is a number. (basic_props:num/1)
– *The following properties hold globally:*
`number_codes(Number,String)` is evaluable at compile-time. (basic_props:eval/1)

atom_number/2: PREDICATE

`atom_number(Atom,Number)`

`Atom` can be read as a representation of `Number`.

General properties: `atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:B=0.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=0.0 (term_basic:= /2)

`atom_number(A,B)`
– *If the following properties should hold at call time:*
term_basic:A=0.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B=0.0 (term_basic:= /2)

`atom_number(A,B)`
– *If the following properties should hold at call time:*
term_basic:B= -0.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=-0.0 (term_basic:= /2)

`atom_number(A,B)`
– *If the following properties should hold at call time:*
term_basic:A=-0.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B0= -0.0 (term_basic:= /2)
term_basic:B=B0 (term_basic:= /2)

`atom_number(A,B)`
– *If the following properties should hold at call time:*
term_basic:B=1.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=1.0 (term_basic:= /2)

`atom_number(A,B)`
– *If the following properties should hold at call time:*
term_basic:A=1.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B=1.0 (term_basic:= /2)

`atom_number(A,B)`
– *If the following properties should hold at call time:*
term_basic:B=0.Inf (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=0.Inf (term_basic:= /2)

`atom_number(A,B)`
– *If the following properties should hold at call time:*
term_basic:A=0.Inf (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B=0.Inf (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:B= -1.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=-1.0 (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:A=-1.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B= -1.0 (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:B= -1.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=-1.0 (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:A=-1.0 (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B= -1.0 (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:B= -0.Inf (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=-0.Inf (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:A=-0.Inf (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B= -0.Inf (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:B= -0.Inf (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=-0.Inf (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:B=0.Nan (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:A=0.Nan (term_basic:= /2)

`atom_number(A,B)`

– *If the following properties should hold at call time:*
term_basic:A=0.Nan (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B=0.Nan (term_basic:= /2)
– *The following properties hold globally:*
`atom_number(Atom,Number)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)
All calls of the form `atom_number(Atom,Number)` are deterministic. (native_props:is_det/1)

Usage 1:
– *Call and exit should be compatible with:*
`Number` is a number. (basic_props:num/1)
– *The following properties should hold at call time:*
`Atom` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
`Number` is a number. (basic_props:num/1)
`Atom` is an atom. (basic_props:atm/1)
`Number` is a number. (basic_props:num/1)
– *The following properties should hold globally:*
`atom_number(Atom,Number)` is evaluable at compile-time. (basic_props:eval/1)

Usage 2:
– *The following properties should hold at call time:*
`Atom` is a free variable. (term_typing:var/1)
`Number` is a number. (basic_props:num/1)
– *The following properties should hold upon exit:*
`Atom` is an atom. (basic_props:atm/1)
`Atom` is an atom. (basic_props:atm/1)
`Number` is a number. (basic_props:num/1)
– *The following properties should hold globally:*
`atom_number(Atom,Number)` is evaluable at compile-time. (basic_props:eval/1)

atom_number/3: PREDICATE

`atom_number(Atom,Base,Number)`

`Atom` can be read as a representation of `Number` in base `Base`.

General properties:
– *The following properties hold globally:*
`atom_number(Atom,Base,Number)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)
All calls of the form `atom_number(Atom,Base,Number)` are deterministic. (native_props:is_det/1)

Usage 1:
– *Call and exit should be compatible with:*
`Base` is a number. (basic_props:num/1)

– *The following properties should hold at call time:*
`Atom` is an atom. (basic_props:atm/1)
`Number` is a number. (basic_props:num/1)
– *The following properties should hold upon exit:*
`Base` is a number. (basic_props:num/1)

Usage 2:
– *The following properties should hold at call time:*
`Atom` is a free variable. (term_typing:var/1)
`Base` is a number. (basic_props:num/1)
`Number` is a number. (basic_props:num/1)
– *The following properties should hold upon exit:*
`Atom` is an atom. (basic_props:atm/1)

atom_length/2: PREDICATE

`atom_length(Atom,Length)` ⟨• ISO •⟩

`Length` is the number of characters forming the name of `Atom`.

General properties:
– *The following properties hold globally:*
`atom_length(Atom,Length)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)
All calls of the form `atom_length(Atom,Length)` are deterministic. (native_props:is_det/1)

Usage:
– *Calls should, and exit will be compatible with:*
`Length` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Atom` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Length` is an integer. (basic_props:int/1)
– *The following properties hold globally:*
`atom_length(Atom,Length)` is evaluable at compile-time. (basic_props:eval/1)

atom_concat/3: PREDICATE

`atom_concat(Atom_1,Atom_2,Atom_12)` ⟨• ISO •⟩

`Atom_12` is the result of concatenating `Atom_1` followed by `Atom_2`.

General properties:
– *The following properties hold globally:*
`atom_concat(Atom_1,Atom_2,Atom_12)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)
All calls of the form `atom_concat(Atom_1,Atom_2,Atom_12)` are deterministic. (native_props:is_det/1)

Usage 1:

– *Description:* Concatenate two atoms.
– *Calls should, and exit will be compatible with:*
`Atom_12` is an atom. (basic_props:atm/1)
– *The following properties should hold at call time:*
`Atom_1` is an atom. (basic_props:atm/1)
`Atom_2` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Atom_12` is an atom. (basic_props:atm/1)
`Atom_1` is an atom. (basic_props:atm/1)
`Atom_2` is an atom. (basic_props:atm/1)
`Atom_12` is an atom. (basic_props:atm/1)
– *The following properties hold globally:*
`atom_concat(Atom_1,Atom_2,Atom_12)` is evaluable at compile-time. (basic_props:eval/1)

Usage 2:
– *Description:* Non-deterministically split an atom.
– *The following properties should hold at call time:*
`Atom_1` is a free variable. (term_typing:var/1)
`Atom_2` is a free variable. (term_typing:var/1)
`Atom_12` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Atom_1` is an atom. (basic_props:atm/1)
`Atom_2` is an atom. (basic_props:atm/1)
`Atom_1` is an atom. (basic_props:atm/1)
`Atom_2` is an atom. (basic_props:atm/1)
`Atom_12` is an atom. (basic_props:atm/1)
– *The following properties hold globally:*
`atom_concat(Atom_1,Atom_2,Atom_12)` is evaluable at compile-time. (basic_props:eval/1)

Usage 3:
– *Description:* Take out of an atom a certain suffix (or fail if it cannot be done).
– *The following properties should hold at call time:*
`Atom_1` is a free variable. (term_typing:var/1)
`Atom_2` is an atom. (basic_props:atm/1)
`Atom_12` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Atom_1` is an atom. (basic_props:atm/1)
`Atom_1` is an atom. (basic_props:atm/1)
`Atom_2` is an atom. (basic_props:atm/1)
`Atom_12` is an atom. (basic_props:atm/1)
– *The following properties hold globally:*
`atom_concat(Atom_1,Atom_2,Atom_12)` is evaluable at compile-time. (basic_props:eval/1)

Usage 4:

– *Description:* Take out of an atom a certain prefix (or fail if it cannot be done).
– *The following properties should hold at call time:*
`Atom_1` is an atom. (basic_props:atm/1)
`Atom_2` is a free variable. (term_typing:var/1)
`Atom_12` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Atom_2` is an atom. (basic_props:atm/1)
`Atom_1` is an atom. (basic_props:atm/1)
`Atom_2` is an atom. (basic_props:atm/1)
`Atom_12` is an atom. (basic_props:atm/1)
– *The following properties hold globally:*
`atom_concat(Atom_1,Atom_2,Atom_12)` is evaluable at compile-time. (basic_props:eval/1)

sub_atom/4: PREDICATE

`sub_atom(Atom,Before,Length,Sub_atom)`

`Sub_atom` is formed with `Length` consecutive characters of `Atom` after the `Before` character. For example, the goal `sub_atom(summer,1,4,umme)` succeeds.

General properties:

– *The following properties hold globally:*
`sub_atom(Atom,Before,Length,Sub_atom)` is side-effect `free`. (basic_props:sideff/2)
This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage:

– *Calls should, and exit will be compatible with:*
`Sub_atom` is an atom. (basic_props:atm/1)
– *The following properties should hold at call time:*
`Atom` is an atom. (basic_props:atm/1)
`Before` is an integer. (basic_props:int/1)
`Length` is an integer. (basic_props:int/1)
– *The following properties hold upon exit:*
`Sub_atom` is an atom. (basic_props:atm/1)
– *The following properties hold globally:*
`sub_atom(Atom,Before,Length,Sub_atom)` is evaluable at compile-time. (basic_props:eval/1)

valid_base/1: REGTYPE

Usage:

– *Description:* Valid numeric base to convert numbers to strings or atoms

19.3 Known bugs and planned improvements (`atomic_basic`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

20 Arithmetic

Author(s): Daniel Cabeza, Manuel Hermenegildo.

Arithmetic is performed by built-in predicates which take as arguments arithmetic expressions (see `arithexpression/1`) and evaluate them. Terms representing arithmetic expressions can be created dynamically, but at the time of evaluation, each variable in an arithmetic expression must be bound to a non-variable expression (the term must be ground). For example, given the code in the first line a possible shell interaction follows:

```
evaluate(Expression, Answer) :- Answer is Expression.

?- _X=24*9, evaluate(_X+6, Ans).

Ans = 222 ?

yes
```

20.1 Usage and interface (`arithmetic`)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `is/2`, `</2`, `=</2`, `>/2`, `>=/2`, `=:=/2`, `=\=/2`.
  - *Regular Types:*
    `arithexpression/1`, `intexpression/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

20.2 Documentation on exports (`arithmetic`)

is/2: PREDICATE

`Val is Exp` ⟨• ISO •⟩

The arithmetic expression `Exp` is evaluated and the result is unified with `Val`

General properties: `X is Y`

- *Description:* is/2, sqrt test
- *If the following properties should hold at call time:*
  term_basic:Y=sqrt(4) (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:X=2.0 (term_basic:= /2)

`A is B`

- *The following properties hold globally:*
  `int` is the metric of the variable `A`, for any approximation. (native_props:size_metric/3)

`int` is the metric of the variable `B`, for any approximation. (native_props:size_metric/3)

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`Val is Exp` is side-effect `free`. (basic_props:sideff/2)

`Val is Exp` is binding insensitive. (basic_props:bind_ins/1)

All calls of the form `Val is Exp` are deterministic. (native_props:is_det/1)

Goal `Val is Exp` produces `inf` solutions. (native_props:relations/2)

Usage 1: `A is B`

– *The following properties should hold at call time:*

`A` is a free variable. (term_typing:var/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is a free variable. (term_typing:var/1)

`B` is an arithmetic expression. (arithmetic:arithexpression/1)

– *The following properties hold upon exit:*

`A` is a number. (basic_props:num/1)

`B` is an arithmetic expression. (arithmetic:arithexpression/1)

`int(B)` is the size of argument `A`, for any approximation. (native_props:size/2)

– *The following properties hold globally:*

All the calls of the form `A is B` do not fail. (native_props:not_fails/1)

`A is B` is evaluable at compile-time. (basic_props:eval/1)

Usage 2: `A is B`

– *The following properties should hold at call time:*

`A` is a free variable. (term_typing:var/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is a free variable. (term_typing:var/1)

`B` is an integer expression. (arithmetic:intexpression/1)

– *The following properties hold upon exit:*

`A` is an integer. (basic_props:int/1)

`B` is an integer expression. (arithmetic:intexpression/1)

`int(B)` is the size of argument `A`, for any approximation. (native_props:size/2)

– *The following properties hold globally:*

All the calls of the form `A is B` do not fail. (native_props:not_fails/1)

`A is B` is evaluable at compile-time. (basic_props:eval/1)

Usage 3: `A is B`

– *The following properties should hold at call time:*

`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is a number. (basic_props:num/1)

`B` is an arithmetic expression. (arithmetic:arithexpression/1)

– *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage 4: `A is B`

– *The following properties should hold at call time:*

`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is an integer. (basic_props:int/1)

`B` is an integer expression. (arithmetic:intexpression/1)

– *The following properties hold globally:*

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

</2: PREDICATE

`Exp1<Exp2` ⟨• ISO •⟩

The numeric value of `Exp1` is less than the numeric value of `Exp2` when both are evaluated as arithmetic expressions.

General properties:

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`Exp1<Exp2` is side-effect `free`. (basic_props:sideff/2)

`Exp1<Exp2` is binding insensitive. (basic_props:bind_ins/1)

All calls of the form `Exp1<Exp2` are deterministic. (native_props:is_det/1)

Goal `Exp1<Exp2` produces `inf` solutions. (native_props:relations/2)

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `A<B`

– *The following properties should hold at call time:*

`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is an arithmetic expression. (arithmetic:arithexpression/1)

`B` is an arithmetic expression. (arithmetic:arithexpression/1)

– *The following properties hold globally:*

`A<B` is evaluable at compile-time. (basic_props:eval/1)

`int` is the metric of the variable `A`, for any approximation. (native_props:size_metric/3)

`int` is the metric of the variable `B`, for any approximation. (native_props:size_metric/3)

=</2: PREDICATE

`Exp1=<Exp2` ⟨• ISO •⟩

The numeric value of `Exp1` is less than or equal to the numeric value of `Exp2` when both are evaluated as arithmetic expressions.

General properties:

– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)
`Exp1=<Exp2` is side-effect `free`. (basic_props:sideff/2)
`Exp1=<Exp2` is binding insensitive. (basic_props:bind_ins/1)
All calls of the form `Exp1=<Exp2` are deterministic. (native_props:is_det/1)
Goal `Exp1=<Exp2` produces `inf` solutions. (native_props:relations/2)
Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `A=<B`

– *The following properties should hold at call time:*
`A` is currently a term which is not a free variable. (term_typing:nonvar/1)
`B` is currently a term which is not a free variable. (term_typing:nonvar/1)
`A` is an arithmetic expression. (arithmetic:arithexpression/1)
`B` is an arithmetic expression. (arithmetic:arithexpression/1)

– *The following properties hold globally:*
`A=<B` is evaluable at compile-time. (basic_props:eval/1)
`int` is the metric of the variable `A`, for any approximation. (native_props:size_metric/3)
`int` is the metric of the variable `B`, for any approximation. (native_props:size_metric/3)

>/2: PREDICATE

`Exp1>Exp2` ⟨• ISO •⟩

The numeric value of `Exp1` is greater than the numeric value of `Exp2` when both are evaluated as arithmetic expressions.

General properties:

– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)
`Exp1>Exp2` is side-effect `free`. (basic_props:sideff/2)
`Exp1>Exp2` is binding insensitive. (basic_props:bind_ins/1)
All calls of the form `Exp1>Exp2` are deterministic. (native_props:is_det/1)
Goal `Exp1>Exp2` produces `inf` solutions. (native_props:relations/2)
Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `A>B`

– *The following properties should hold at call time:*
`A` is currently a term which is not a free variable. (term_typing:nonvar/1)
`B` is currently a term which is not a free variable. (term_typing:nonvar/1)
`A` is an arithmetic expression. (arithmetic:arithexpression/1)
`B` is an arithmetic expression. (arithmetic:arithexpression/1)

– *The following properties hold globally:*
`A>B` is evaluable at compile-time. (basic_props:eval/1)
`int` is the metric of the variable `A`, for any approximation. (native_props:size_metric/3)
`int` is the metric of the variable `B`, for any approximation. (native_props:size_metric/3)

>=/2: PREDICATE

`Exp1>=Exp2` ⟨• ISO •⟩

The numeric value of `Exp1` is greater than or equal to the numeric value of `Exp2` when both are evaluated as arithmetic expressions.

General properties:

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`Exp1>=Exp2` is side-effect `free`. (basic_props:sideff/2)

`Exp1>=Exp2` is binding insensitive. (basic_props:bind_ins/1)

All calls of the form `Exp1>=Exp2` are deterministic. (native_props:is_det/1)

Goal `Exp1>=Exp2` produces `inf` solutions. (native_props:relations/2)

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `A>=B`

– *The following properties should hold at call time:*

`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is an arithmetic expression. (arithmetic:arithexpression/1)

`B` is an arithmetic expression. (arithmetic:arithexpression/1)

– *The following properties hold globally:*

`A>=B` is evaluable at compile-time. (basic_props:eval/1)

`int` is the metric of the variable `A`, for any approximation. (native_props:size_metric/3)

`int` is the metric of the variable `B`, for any approximation. (native_props:size_metric/3)

=:=/2: PREDICATE

`Exp1=:=Exp2` ⟨• ISO •⟩

The numeric values of `Exp1` and `Exp2` are equal when both are evaluated as arithmetic expressions.

General properties:

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`Exp1=:=Exp2` is side-effect `free`. (basic_props:sideff/2)

`Exp1=:=Exp2` is binding insensitive. (basic_props:bind_ins/1)

All calls of the form `Exp1=:=Exp2` are deterministic. (native_props:is_det/1)

Goal `Exp1=:=Exp2` produces `inf` solutions. (native_props:relations/2)

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `A=:=B`

– *The following properties should hold at call time:*

`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is an arithmetic expression. (arithmetic:arithexpression/1)

`B` is an arithmetic expression. (arithmetic:arithexpression/1)

– *The following properties hold globally:*

`A=:=B` is evaluable at compile-time. (basic_props:eval/1)

`int` is the metric of the variable `A`, for any approximation. (native_props:size_metric/3)

`int` is the metric of the variable `B`, for any approximation. (native_props:size_metric/3)

=\=/2: PREDICATE

`Exp1=\=Exp2` ⟨• ISO •⟩

The numeric values of `Exp1` and `Exp2` are not equal when both are evaluated as arithmetic expressions.

General properties:

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

`Exp1=\=Exp2` is side-effect `free`. (basic_props:sideff/2)

`Exp1=\=Exp2` is binding insensitive. (basic_props:bind_ins/1)

All calls of the form `Exp1=\=Exp2` are deterministic. (native_props:is_det/1)

Goal `Exp1=\=Exp2` produces `inf` solutions. (native_props:relations/2)

Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `A=\=B`

– *The following properties should hold at call time:*

`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`B` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is an arithmetic expression. (arithmetic:arithexpression/1)

`B` is an arithmetic expression. (arithmetic:arithexpression/1)

– *The following properties hold globally:*

`A=\=B` is evaluable at compile-time. (basic_props:eval/1)

`int` is the metric of the variable `A`, for any approximation. (native_props:size_metric/3)

`int` is the metric of the variable `B`, for any approximation. (native_props:size_metric/3)

arithexpression/1: REGTYPE

An arithmetic expression is a term built from numbers and evaluable functors that represent arithmetic functions. An arithmetic expression evaluates to a number, which may be an integer (`int/1`) or a float (`flt/1`). The evaluable functors allowed in an arithmetic expression are listed below, together with an indication of the functions they represent. All evaluable functors defined in ISO-Prolog are implemented, as well as some other useful or traditional. Unless stated otherwise, an expression evaluates to a float if any of its arguments is a float, otherwise to an integer.

- `- /1`: sign reversal. ⟨• ISO •⟩
- `+ /1`: identity.
- `-- /1`: decrement by one.

- `++ /1`: increment by one.
- `+ /2`: addition. ⟨• ISO •⟩
- `- /2`: subtraction. ⟨• ISO •⟩
- `* /2`: multiplication. ⟨• ISO •⟩
- `// /2`: integer division. Float arguments are truncated to integers, result always integer. ⟨• ISO •⟩
- `/ /2`: division. Result always float. ⟨• ISO •⟩
- `rem/2`: integer remainder. The result is always an integer, its sign is the sign of the first argument. ⟨• ISO •⟩
- `mod/2`: modulo. The result is always a positive integer. ⟨• ISO •⟩
- `abs/1`: absolute value. ⟨• ISO •⟩
- `sign/1`: sign of. ⟨• ISO •⟩
- `float_integer_part/1`: float integer part. Result always float. ⟨• ISO •⟩
- `float_fractional_part/1`: float fractional part. Result always float. ⟨• ISO •⟩
- `truncate/1`: The result is the integer equal to the integer part of the argument. ⟨• ISO •⟩
- `integer/1`: same as `truncate/1`.
- `float/1`: conversion to float. ⟨• ISO •⟩
- `floor/1`: largest integer not greater than. ⟨• ISO •⟩
- `round/1`: integer nearest to. ⟨• ISO •⟩
- `ceiling/1`: smallest integer not smaller than. ⟨• ISO •⟩
- `** /2`: exponentiation. Result always float. ⟨• ISO •⟩
- `>> /2`: integer bitwise right shift. ⟨• ISO •⟩
- `<< /2`: integer bitwise left shift. ⟨• ISO •⟩
- `/\ /2`: integer bitwise and. ⟨• ISO •⟩
- `\/ /2`: integer bitwise or. ⟨• ISO •⟩
- `\ /1`: integer bitwise complement. ⟨• ISO •⟩
- `# /2`: integer bitwise exclusive or (xor).
- `exp/1`: exponential (e to the power of). Result always float. ⟨• ISO •⟩
- `log/1`: natural logarithm (base e). Result always float. ⟨• ISO •⟩
- `sqrt/1`: square root. Result always float. ⟨• ISO •⟩
- `sin/1`: sine. Result always float. ⟨• ISO •⟩
- `cos/1`: cosine. Result always float. ⟨• ISO •⟩
- `atan/1`: arc tangent. Result always float. ⟨• ISO •⟩
- `gcd/2`: Greatest common divisor. Arguments must evaluate to integers, result always integer.

In addition to these functors, a list of just a number evaluates to this number. Since a quoted string is just a list of integers, this allows a quoted character to be used in place of its ASCII code; e.g. `"A"` behaves within arithmetic expressions as the integer 65. Note that this is not ISO-compliant, and that can be achieved by using the ISO notation `0'A`.

Arithmetic expressions, as described above, are just data structures. If you want one evaluated you must pass it as an argument to one of the arithmetic predicates defined in this library.

General properties:

– *The following properties hold globally:*
`arithexpression(Arg1)` is side-effect `free`. (basic_props:sideff/2)
– *If the following properties hold at call time:*
`Arg1` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
All calls of the form `arithexpression(Arg1)` are deterministic. (native_props:is_det/1)
Indicates the type of test that a predicate performs. Required by the nonfailure analyisis. (native_props:test_type/2)

Usage: `arithexpression(E)`
– *Description:* `E` is an arithmetic expression.

intexpression/1: REGTYPE
General properties:
– *The following properties hold globally:*
`intexpression(Arg1)` is side-effect `free`. (basic_props:sideff/2)

Usage: `intexpression(E)`
– *Description:* `E` is an integer expression.

20.3 Known bugs and planned improvements (`arithmetic`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

21 Basic file/stream handling

Author(s): Daniel Cabeza, Mats Carlsson.

This module provides basic predicates for handling files and streams, in order to make input/output on them.

21.1 Usage and interface (streams_basic)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    open/3, open/4, close/1, set_input/1, current_input/1, set_output/1, current_output/1, character_count/2, line_count/2, line_position/2, flush_output/1, flush_output/0, clearerr/1, current_stream/3, stream_code/2, absolute_file_name/2, absolute_file_name/7, pipe/2.
  - *Regular Types:*
    open_option_list/1, sourcename/1, stream/1, stream_alias/1, io_mode/1, atm_or_int/1.
  - *Multifiles:*
    file_search_path/2, library_directory/1.

21.2 Documentation on exports (streams_basic)

open/3: PREDICATE

`open(File,Mode,Stream)`

Open `File` with mode `Mode` and return in `Stream` the stream associated with the file. No extension is implicit in `File`.

Usage 1: ⟨• ISO •⟩

- *Description:* Normal use.
- *Call and exit should be compatible with:*
  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties should hold at call time:*
  `File` is a source name. (streams_basic:sourcename/1)
  `Mode` is an opening mode ('read', 'write' or 'append'). (streams_basic:io_mode/1)
- *The following properties should hold upon exit:*
  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

Usage 2:

- *Description:* In the special case that `File` is an integer, it is assumed to be a file descriptor passed to Prolog from a foreign function call. The file descriptor is connected to a Prolog stream (invoking the UNIX function `fdopen`) which is unified with `Stream`.

– *Call and exit should be compatible with:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties should hold at call time:*
`File` is an integer. (basic_props:int/1)
`Mode` is an opening mode ('read', 'write' or 'append'). (streams_basic:io_mode/1)
– *The following properties should hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)

open/4: PREDICATE

`open(File,Mode,Stream,Options)`

Same as `open(File, Mode, Stream)` with options `Options`. See the definition of `open_option_list/1` for details.

Usage:

– *Call and exit should be compatible with:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties should hold at call time:*
`File` is a source name. (streams_basic:sourcename/1)
`Mode` is an opening mode ('read', 'write' or 'append'). (streams_basic:io_mode/1)
`Options` is a list of options for `open/4`. (streams_basic:open_option_list/1)
– *The following properties should hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

open_option_list/1: REGTYPE

A list of options for `open/4`, currently the meaningful options are:

`lock` Try to set an advisory lock for the file. If the open mode is `read`, the lock is a read (shared) lock, else it is a write (exclusive) lock. If the lock cannot be acquired, the call waits until it is released (but can fail in exceptional cases).

`lock_nb` Same as `lock`, but the call immediately fails if the lock cannot be acquired.

`lock(Lock_Mode)`
Same as `lock`, but specifying in `Lock_Mode` whether the lock is `read` (also `shared`) or `write` (also `exclusive`). This option has be included for compatibility with the SWI-Prolog locking options, because in general the type of lock should match the open mode as in the `lock` option.

`lock_nb(Lock_Mode)`
Same as the previous option but with the `lock_nb` behavior.

All file locking is implemented via the POSIX function fcntl(). Please refer to its manual page for details.

Usage: `open_option_list(L)`

– *Description:* `L` is a list of options for `open/4`.

close/1: PREDICATE

`close(Stream)`

Close the stream `Stream`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

set_input/1: PREDICATE

`set_input(Stream)`

Set the current input stream to `Stream`. A notion of *current input stream* is maintained by the system, so that input predicates with no explicit stream operate on the current input stream. Initially it is set to `user_input`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_input/1: PREDICATE

`current_input(Stream)`

Unify `Stream` with the current input stream. In addition to the ISO behavior, stream aliases are allowed. This is useful for most applications checking whether a stream is the standard input or output.

Usage: ⟨• ISO •⟩

– *Calls should, and exit will be compatible with:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

set_output/1: PREDICATE

`set_output(Stream)`

Set the current output stream to `Stream`. A notion of *current output stream* is maintained by the system, so that output predicates with no explicit stream operate on the current output stream. Initially it is set to `user_output`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_output/1: PREDICATE

`current_output(Stream)`

Unify `Stream` with the current output stream. The same comment as for `current_input/1` applies.

Usage: ⟨• ISO •⟩

– *Calls should, and exit will be compatible with:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

character_count/2: PREDICATE

`character_count(Stream,Count)`

`Count` characters have been read from or written to `Stream`.

Usage:

– *Calls should, and exit will be compatible with:*
`Count` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`Count` is an integer. (basic_props:int/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

line_count/2: PREDICATE

`line_count(Stream,Count)`

`Count` lines have been read from or written to `Stream`.

Usage:

– *Calls should, and exit will be compatible with:*
`Count` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`Count` is an integer. (basic_props:int/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

line_position/2: PREDICATE

`line_position(Stream,Count)`

`Count` characters have been read from or written to the current line of `Stream`.

Usage:

– *Calls should, and exit will be compatible with:*
`Count` is an integer. (basic_props:int/1)

– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)

– *The following properties hold upon exit:*
`Count` is an integer. (basic_props:int/1)

– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

flush_output/1: PREDICATE

`flush_output(Stream)`

Flush any buffered data to output stream `Stream`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)

– *The following properties hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)

– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

flush_output/0: PREDICATE

`flush_output`

Behaves like `current_output(S), flush_output(S)`

clearerr/1: PREDICATE

`clearerr(Stream)`

Clear the end-of-file and error indicators for input stream `Stream`.

Usage:

– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)

– *The following properties hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)

current_stream/3: PREDICATE

`current_stream(Filename,Mode,Stream)`

`Stream` is a stream which was opened in mode `Mode` and which is connected to the absolute file name `Filename` (an atom) or to the file descriptor `Filename` (an integer). This predicate can be used for enumerating all currently open streams through backtracking.

Usage:

– *Calls should, and exit will be compatible with:*
streams_basic:atm_or_int(Filename) (streams_basic:atm_or_int/1)
`Mode` is an opening mode ('read', 'write' or 'append'). (streams_basic:io_mode/1)
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
streams_basic:atm_or_int(Filename) (streams_basic:atm_or_int/1)
`Mode` is an opening mode ('read', 'write' or 'append'). (streams_basic:io_mode/1)
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

stream_code/2: PREDICATE

`stream_code(Stream,StreamCode)`

`StreamCode` is the file descriptor (an integer) corresponding to the Prolog stream `Stream`.

Usage 1:

– *Calls should, and exit will be compatible with:*
`StreamCode` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`StreamCode` is an integer. (basic_props:int/1)

Usage 2:

– *The following properties should hold at call time:*
`Stream` is a free variable. (term_typing:var/1)
`StreamCode` is an integer. (basic_props:int/1)
– *The following properties hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)

absolute_file_name/2: PREDICATE

`absolute_file_name(RelFileSpec,AbsFileSpec)`

If `RelFileSpec` is an absolute pathname then do an absolute lookup. If `RelFileSpec` is a relative pathname then prefix the name with the name of the current directory and do an absolute lookup. If `RelFileSpec` is a path alias, perform the lookup following the path alias rules (see `sourcename/1`). In all cases: if a matching file with suffix `.pl` exists, then `AbsFileSpec` will be unified with this file. Failure to open a file normally causes an exception. The behaviour can be controlled by the `fileerrors` prolog flag.

Usage: `absolute_file_name(RelFileSpec,AbsFileSpec)`

– *Description:* `AbsFileSpec` is the absolute name (with full path) of `RelFileSpec`.
– *Call and exit should be compatible with:*
`RelFileSpec` is a source name. (streams_basic:sourcename/1)
`AbsFileSpec` is an atom. (basic_props:atm/1)
– *The following properties should hold at call time:*
`RelFileSpec` is currently a term which is not a free variable. (term_typing:nonvar/1)
`AbsFileSpec` is a free variable. (term_typing:var/1)

– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

absolute_file_name/7: PREDICATE

`absolute_file_name(Spec,Opt,Suffix,CurrDir,AbsFile,AbsBase,AbsDir)`

`AbsFile` is the absolute name (with full path) of `Spec`, which has an optional first suffix `Opt` and an optional second suffix `Suffix`, when the current directory is `CurrDir`. `AbsBase` is the same as `AbsFile`, but without the second suffix, and `AbsDir` is the absolute path of the directory where `AbsFile` is. The Ciao compiler invokes this predicate with `Opt='_opt'` and `Suffix='.pl'` when searching source files.

Usage:

– *The following properties should hold at call time:*
`Spec` is a source name. (streams_basic:sourcename/1)
`Opt` is an atom. (basic_props:atm/1)
`Suffix` is an atom. (basic_props:atm/1)
`CurrDir` is an atom. (basic_props:atm/1)
`AbsFile` is a free variable. (term_typing:var/1)
`AbsBase` is a free variable. (term_typing:var/1)
`AbsDir` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`AbsFile` is an atom. (basic_props:atm/1)
`AbsBase` is an atom. (basic_props:atm/1)
`AbsDir` is an atom. (basic_props:atm/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

pipe/2: PREDICATE

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*
`Arg1` is a free variable. (term_typing:var/1)
`Arg2` is a free variable. (term_typing:var/1)
– *The following properties hold upon exit:*
`Arg1` is an open stream. (streams_basic:stream/1)
`Arg2` is an open stream. (streams_basic:stream/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

sourcename/1: REGTYPE

A source name is a flexible way of referring to a concrete file. A source name is either a relative or absolute filename given as:

- an atom, or
- a unary functor (which represents a *path alias*, see below) applied to a *relative* path, the latter being given as an atom.

In all cases certain filename extensions (e.g., `.pl`) can be implicit. In the first form above, file names can be relative to the current directory. Also, file names beginning with `~` or `$` are treated specially. For example,

`'~/ciao/sample.pl'`
: is equivalent to `'/home/staff/herme/ciao/sample.pl'`, if `/home/staff/herme` is the user's home directory. (This is also equivalent to `'$HOME/ciao/sample.pl'` as explained below.)

`'~bardo/prolog/sample.pl'`
: is equivalent to `'/home/bardo/prolog/sample.pl'`, if `/home/bardo` is bardo's home directory.

`'$UTIL/sample.pl'`
: is equivalent to `'/usr/local/src/utilities/sample.pl'`, if `/usr/local/src/utilities` is the value of the environment variable `UTIL`.

The second form allows using path aliases. Such aliases allow refering to files not with absolute file system paths but with paths which are relative to predefined (or user-defined) abstract names. For example, given the path alias `myutils` which has been defined to refer to path `'/home/bardo/utilities'`, if that directory contains the file `stuff.pl` then the term `myutils(stuff)` in a `use_module/1` declaration would refer to the file `'/home/bardo/utilities/stuff.pl'` (the `.pl` extension is implicit in the `use_module/1` declaration). As a special case, if that directory contains a subdirectory named `stuff` which in turn contains the file `stuff.pl`, the same term would refer to the file `'/home/bardo/utilities/stuff/stuff.pl'`. If a path alias is related to several paths, all paths are scanned in sequence until a match is found. For information on predefined path aliases or how to define new path aliases, see `file_search_path/2`.

Usage: `sourcename(F)`

– *Description:* `F` is a source name.

stream/1: REGTYPE

Streams correspond to the file pointers used at the operating system level, and usually represent opened files. There are four special streams which correspond with the operating system standard streams:

`user_input`
: The standard input stream, i.e. the terminal, usually.

`user_output`
: The standard output stream, i.e. the terminal, usually.

`user_error`
: The standard error stream.

`user` The standard input or output stream, depending on context.

Usage: `stream(S)`

– *Description:* `S` is an open stream.

stream_alias/1: REGTYPE

Usage: `stream_alias(S)`

– *Description:* `S` is the alias of an open stream, i.e., an atom which represents a stream at Prolog level.

io_mode/1: REGTYPE

Can have the following values:

`read` Open the file for input.

`write` Open the file for output. The file is created if it does not already exist, the file will otherwise be truncated.

`append` Open the file for output. The file is created if it does not already exist, the file will otherwise be appended to.

Usage: `io_mode(M)`

– *Description:* `M` is an opening mode ('read', 'write' or 'append').

atm_or_int/1: REGTYPE

A regular type, defined as follows:

```
atm_or_int(X) :-
        atm(X).
atm_or_int(X) :-
        int(X).
```

21.3 Documentation on multifiles (`streams_basic`)

file_search_path/2: PREDICATE

`file_search_path(Alias,Path)`

The path alias `Alias` is linked to path `Path`. Both arguments must be atoms. New facts (or clauses) of this predicate can be asserted to define new path aliases. Predefined path aliases in Ciao are:

`library` Initially points to all Ciao library paths. See `library_directory/1`.

`engine` The path of the Ciao engine builtins.

`.` The current path (`'.'`).

The predicate is *multifile*.

The predicate is of type *dynamic*.

`file_search_path(X,Y)`

– *The following properties hold upon exit:*

`X` is ground. (basic_props:gnd/1)

`Y` is ground. (basic_props:gnd/1)

library_directory/1: PREDICATE

`library_directory(Path)`

`Path` is a library path (a path represented by the path alias `library`). Predefined library paths in Ciao are `'$CIAOLIB/lib'`, `'$CIAOLIB/library'`, and `'$CIAOLIB/contrib'`, given that `$CIAOLIB` is the path of the root ciao library directory. More library paths can be defined by asserting new facts (or clauses) of this predicate.

The predicate is *multifile*.

The predicate is of type *dynamic*.

`library_directory(X)`

– *The following properties hold upon exit:*
`X` is ground. (basic_props:gnd/1)

21.4 Known bugs and planned improvements (`streams_basic`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

22 Basic input/output

Author(s): Daniel Cabeza, Mats Carlsson.

This module provides predicates for character input/output and for canonical term output. From the ISO-Prolog predicates for character input/output, only the `_code` versions are provided, the rest are given by `library(iso_byte_char)`, using these. Most predicates are provided in two versions: one that specifies the input or output stream as the first argument and a second which omits this argument and uses the current input or output stream.

22.1 Usage and interface (`io_basic`)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `get_code/2`, `get_code/1`, `get1_code/2`, `get1_code/1`, `peek_code/2`, `peek_code/1`, `skip_code/2`, `skip_code/1`, `skip_line/1`, `skip_line/0`, `put_code/2`, `put_code/1`, `nl/1`, `nl/0`, `tab/2`, `tab/1`, `code_class/2`, `getct/2`, `getct1/2`, `display/2`, `display/1`, `displayq/2`, `displayq/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

22.2 Documentation on exports (`io_basic`)

get_code/2: PREDICATE

`get_code(Stream,Code)`

Reads from `Stream` the next character and unifies `Code` with its character code. At end of stream, unifies `Code` with the integer -1.

Usage: ⟨• ISO •⟩

- *Calls should, and exit will be compatible with:*
  `Code` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*
  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties hold upon exit:*
  `Code` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

get_code/1: PREDICATE

`get_code(Code)`

Behaves like `current_input(S), get_code(S,Code)`.

Usage: ⟨• ISO •⟩

– *Calls should, and exit will be compatible with:*
`Code` is an integer. (basic_props:int/1)
– *The following properties hold upon exit:*
`Code` is an integer. (basic_props:int/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

get1_code/2: PREDICATE

`get1_code(Stream,Code)`

Reads from `Stream` the next non-layout character (see `code_class/2`) and unifies `Code` with its character code. At end of stream, unifies `Code` with the integer -1.

Usage:

– *Calls should, and exit will be compatible with:*
`Code` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`Code` is an integer. (basic_props:int/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

get1_code/1: PREDICATE

`get1_code(Code)`

Behaves like `current_input(S), get1_code(S,Code)`.

Usage:

– *Calls should, and exit will be compatible with:*
`Code` is an integer. (basic_props:int/1)
– *The following properties hold upon exit:*
`Code` is an integer. (basic_props:int/1)
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

peek_code/2: PREDICATE

`peek_code(Stream,Code)`

Unifies `Code` with the character code of the next character of `Stream`, leaving the stream position unaltered. At end of stream, unifies `Code` with the integer -1.

Usage: ⟨• ISO •⟩

– *Calls should, and exit will be compatible with:*
`Code` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties hold upon exit:*
`Code` is an integer. (basic_props:int/1)

peek_code/1: PREDICATE

`peek_code(Code)`

Behaves like `current_input(S), peek_code(S,Code)`.

Usage: ⟨• ISO •⟩

– *Calls should, and exit will be compatible with:*

`Code` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Code` is an integer. (basic_props:int/1)

skip_code/2: PREDICATE

`skip_code(Stream,Code)`

Skips just past the next character code `Code` from `Stream`.

Usage:

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

`Code` is an integer. (basic_props:int/1)

skip_code/1: PREDICATE

`skip_code(Code)`

Behaves like `current_input(S), skip_code(S,Code)`.

Usage:

– *The following properties should hold at call time:*

`Code` is an integer. (basic_props:int/1)

skip_line/1: PREDICATE

`skip_line(Stream)`

Skips from `Stream` the remaining input characters on the current line. If the end of the stream is reached, the stream will stay at its end. Portable among different operating systems.

Usage:

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

skip_line/0: PREDICATE

`skip_line`

Behaves like `current_input(S), skip_line(S)`.

put_code/2: PREDICATE

`put_code(Stream,Code)`

Outputs to `Stream` the character corresponding to character code `Code`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

`Code` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

All calls of the form `put_code(Stream,Code)` are deterministic. (native_props:is_det/1)

put_code/1: PREDICATE

`put_code(Code)`

Behaves like `current_output(S), put_code(S,Code)`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*

`Code` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

All calls of the form `put_code(Code)` are deterministic. (native_props:is_det/1)

nl/1: PREDICATE

`nl(Stream)`

Outputs a newline character to `Stream`. Equivalent to `put_code(Stream, 0'\n)`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

All calls of the form `nl(Stream)` are deterministic. (native_props:is_det/1)

All the calls of the form `nl(Stream)` do not fail. (native_props:not_fails/1)

nl/0: PREDICATE

`nl`

Behaves like `current_output(S), nl(S)`.

Usage: ⟨• ISO •⟩

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

All calls of the form `nl` are deterministic. (native_props:is_det/1)

All the calls of the form `nl` do not fail. (native_props:not_fails/1)

Goal `nl` produces `1` solutions. (native_props:relations/2)

tab/2: PREDICATE

`tab(Stream,Num)`

Outputs `Num` spaces to `Stream`.

Usage:

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

`Num` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

All calls of the form `tab(Stream,Num)` are deterministic. (native_props:is_det/1)

tab/1: PREDICATE

`tab(Num)`

Behaves like `current_output(S), tab(S,Num)`.

Usage:

– *The following properties should hold at call time:*

`Num` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

All calls of the form `tab(Num)` are deterministic. (native_props:is_det/1)

code_class/2: PREDICATE

`code_class(Code,Class)`

Unifies `Class` with an integer corresponding to the lexical class of the character whose code is `Code`, with the following correspondence:

```
0 - layout (includes space, newline, tab)
1 - small letter
2 - capital letter (including '_')
3 - digit
4 - graphic (includes #$&*+-./:<=>?@^\`~ )
5 - punctuation (includes !;"'%(),[]{|} )
```

Note that in ISO-Prolog the back quote ` is a punctuation character, whereas in Ciao it is a graphic character. Thus, if compatibility with ISO-Prolog is desired, the programmer should not use this character in unquoted names.

Usage:

– *Calls should, and exit will be compatible with:*

`Class` is an integer. (basic_props:int/1)

– *The following properties should hold at call time:*

`Code` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Class` is an integer. (basic_props:int/1)

getct/2: PREDICATE

`getct(Code,Type)`

Reads from the current input stream the next character, unifying `Code` with its character code, and `Type` with its lexical class. At end of stream, unifies both `Code` and `Type` with the integer -1. Equivalent to

```
get(Code), (Code = -1 -> Type = -1 ; code_class(Code,Type))
```

Usage:

– *Calls should, and exit will be compatible with:*

`Code` is an integer. (basic_props:int/1)

`Type` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Code` is an integer. (basic_props:int/1)

`Type` is an integer. (basic_props:int/1)

getct1/2: PREDICATE

`getct1(Code,Type)`

Reads from the current input stream the next non-layout character, unifying `Code` with its character code, and `Type` with its lexical class (which will be nonzero). At end of stream, unifies both `Code` and `Type` with the integer -1. Equivalent to

```
get1(Code), (Code = -1 -> Type = -1 ; code_class(Code,Type))
```

Usage:

– *Calls should, and exit will be compatible with:*

`Code` is an integer. (basic_props:int/1)

`Type` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Code` is an integer. (basic_props:int/1)

`Type` is an integer. (basic_props:int/1)

display/2: PREDICATE

`display(Stream,Term)`

Displays `Term` onto `Stream`. Lists are output using list notation, the other compound terms are output in functional notation. Similar to `write_term(Stream, Term, [ignore_ops(ops)])`, except that curly bracketed notation is not used with `{}/1`, and the `write_strings` flag is not honored.

Usage:

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

`Term` is any term. (basic_props:term/1)

– *The following properties hold upon exit:*

`Term` is any term. (basic_props:term/1)

– *The following properties hold globally:*

`Term` is not further instantiated. (basic_props:not_further_inst/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

display/1: PREDICATE

`display(Term)`

Behaves like `current_output(S), display(S,Term)`.

Usage:

– *The following properties should hold at call time:*

`Term` is any term. (basic_props:term/1)

– *The following properties hold upon exit:*

`Term` is any term. (basic_props:term/1)

– *The following properties hold globally:*

`Term` is not further instantiated. (basic_props:not_further_inst/2)

This predicate is understood natively by CiaoPP. (basic_props:native/1)

displayq/2: PREDICATE

`displayq(Stream,Term)`

Similar to `display(Stream, Term)`, but atoms and functors that can't be read back by `read_term/3` are quoted. Thus, similar to `write_term(Stream, Term, [quoted(true), ignore_ops(ops)])`, with the same exceptions as `display/2`.

Usage:

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

`Term` is any term. (basic_props:term/1)

– *The following properties hold upon exit:*

`Term` is any term. (basic_props:term/1)

– *The following properties hold globally:*

`Term` is not further instantiated. (basic_props:not_further_inst/2)

displayq/1: PREDICATE

`displayq(Term)`

Behaves like `current_output(S), displayq(S,Term)`.

Usage:

– *The following properties should hold at call time:*

`Term` is any term. (basic_props:term/1)

– *The following properties hold upon exit:*

`Term` is any term. (basic_props:term/1)

– *The following properties hold globally:*

`Term` is not further instantiated. (basic_props:not_further_inst/2)

22.3 Known bugs and planned improvements (`io_basic`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

23 Exception and Signal handling

Author(s): The CLIP Group.

This module includes predicates related to exceptions and signals, which alter the normal flow of Prolog.

23.1 Usage and interface (exceptions)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `catch/3`, `intercept/3`, `throw/1`, `send_signal/1`, `send_silent_signal/1`, `halt/0`, `halt/1`, `abort/0`.

23.2 Documentation on exports (exceptions)

catch/3: PREDICATE

`catch(Goal,Error,Handler)`

Executes `Goal`. If an exception is raised during its execution, `Error` is unified with the exception, and if the unification succeeds, the entire execution derived from `Goal` is aborted, and `Handler` is executed. The execution resumes with the continuation of the `catch/3` call. For example, given the code

```
p(X) :- throw(error), display('---').
p(X) :- display(X).
```

the execution of `"catch(p(0), E, display(E)), display(.), fail."` results in the output `"error."`.

Meta-predicate with arguments: `catch(goal,?,goal)`.

Usage: ⟨• ISO •⟩

- *Calls should, and exit will be compatible with:*
  `Error` is any term. (basic_props:term/1)
  `Handler` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties should hold at call time:*
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties hold upon exit:*
  `Error` is any term. (basic_props:term/1)
  `Handler` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

intercept/3: PREDICATE

`intercept(Goal,Signal,Handler)`

Executes `Goal`. If a signal is send during its execution, `Signal` is unified with the exception, and if the unification succeeds, `Handler` is executed and then the execution resumes after the point where the exception was thrown. To avoid infinite loops if `Handler` raises an exception which unifies with `Error`, the exception handler is deactivated before to execute `Handler`. Note the difference with builtin `catch/3`, given the code

```
p(X) :- send_signal(signal), display('---').
p(X) :- display(X).
```

the execution of `"intercept(p(0), E, display(E)), display(.), fail."` results in the output `"error---.0."`.

Meta-predicate with arguments: `intercept(goal,?,goal)`.

Usage:

– *Call and exit should be compatible with:*

`Signal` is any term. (basic_props:term/1)

– *The following properties should hold at call time:*

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Handler` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Signal` is any term. (basic_props:term/1)

throw/1: PREDICATE

`throw(Ball)`

Raises an error, throwing the exception `Ball`, to be caught by an ancestor `catch/3`. The closest matching ancestor is chosen. Exceptions are also thrown by other builtins in case of error.

Usage: `throw(Term)` ⟨• ISO •⟩

– *The following properties should hold at call time:*

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

send_signal/1: PREDICATE

`send_signal(Signal)`

Emits a signal, to be intercept by an ancestor `intercept/3`. The closest matching ancestor is chosen. If the signal is not intercepted, throws the error `error(unintercepted_signal(Signal), send_signal/1-1)`.

Usage: `send_signal(Term)`

– *The following properties should hold at call time:*

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

send_silent_signal/1: PREDICATE

`send_silent_signal(Signal)`

Emits a signal as `send_signal/1`, but do not throws an error if the signal is not intercepted (i.e. just suceeds silently)

Usage: `send_silent_signal(Term)`

– *The following properties should hold at call time:*

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

halt/0: PREDICATE

`halt`

Halt the system, exiting to the invoking shell.

Usage: ⟨• ISO •⟩

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

halt/1: PREDICATE

`halt(Code)`

Halt the system, exiting to the invoking shell, returning exit code `Code`.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*

`Code` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`Code` is an integer. (basic_props:int/1)

abort/0: PREDICATE

`abort`

Abort the current execution.

23.3 Known bugs and planned improvements (`exceptions`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

24 Changing system behaviour and various flags

Author(s): Daniel Cabeza, Mats Carlsson.

Flags define some parameters of the system and control the behavior of system or library predicates. Each flag has a name and an associated predefined value, and except some system flags which are fixed in general their associated value is changeable. Predefined flags in the system are:

`version` The Ciao version, as a term `ciao(Version,Patch)`. `Version` is a floating point number, `Patch` is an integer. Unchangeable.

`dialect` Value set to `ciao`. Used for compatibility with other systems when in Prolog mode. Unchangeable.

`argv` Its value is a list of atoms representing the program arguments supplied when the current executable was invoked. This is the value to which is instantiated the argument of the `main/1` predicate at executable startup. Unchangeable.

`bounded` It is `false`, to denote that the range of integers can be considered infinite (but see `int/1`). Unchangeable. ⟨• ISO •⟩

`fileerrors`
If `on`, predicates handling files give errors (throw exceptions) when a file is inexistent or an operation is not allowed. If `off`, fail in that conditions. Initially `on`.

`gc` Controls whether garbage collection is done. May be `on` (default) or `off`.

`gc_margin`
An integer `Margin`. If less than `Margin` kilobytes are reclaimed in a garbage collection then the size of the garbage collected area should be increased. Also, no garbage collection is attempted unless the garbage collected area has at least `Margin` kilobytes. Initially 500.

`gc_trace` Governs garbage collection trace messages. An element off `[on,off,terse,verbose]`. Initially `off`.

`integer_rounding_function`
It is `toward_zero`, so that `-1 =:= -3//2` succeeds. Unchangeable. ⟨• ISO •⟩

`max_arity`
It is 255, so that no compound term (or predicate) can have more than this number of arguments. Unchangeable. ⟨• ISO •⟩

`quiet` Controls which messages issued using `io_aux` are actually written. As the system uses that library to report its messages, this flag controls the *verbosity* of the system. Possible states of the flag are:

- `on` No messages are reported.
- `error` Only error messages are reported.
- `warning` Only error and warning messages are reported.
- `off` All messages are reported, except debug messages. This is the default state.
- `debug` All messages, including debug messages, are reported. This is only intended for the system implementators.

`unknown` Controls action on calls to undefined predicates. The possible states of the flag are:

- `error` An error is thrown with the error term `existence_error(procedure, F/A)`.

`fail` The call simply fails.

`warning` A warning is written and the call fails.

The state is initially `error`. ⟨• ISO •⟩

24.1 Usage and interface (`prolog_flags`)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `set_prolog_flag/2`, `current_prolog_flag/2`, `prolog_flag/3`, `push_prolog_flag/2`, `pop_prolog_flag/1`, `set_ciao_flag/2`, `current_ciao_flag/2`, `ciao_flag/3`, `push_ciao_flag/2`, `pop_ciao_flag/1`, `prompt/2`, `gc/0`, `nogc/0`, `fileerrors/0`, `nofileerrors/0`.
  - *Multifiles:*
    `define_flag/3`.

24.2 Documentation on exports (`prolog_flags`)

set_prolog_flag/2: PREDICATE

`set_prolog_flag(FlagName,Value)`

Set existing flag `FlagName` to `Value`.

Usage: ⟨• ISO •⟩

- *The following properties should hold at call time:*
  `FlagName` is an atom. (basic_props:atm/1)
  `Value` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*
  `FlagName` is an atom. (basic_props:atm/1)
  `Value` is any term. (basic_props:term/1)

current_prolog_flag/2: PREDICATE

`current_prolog_flag(FlagName,Value)`

`FlagName` is an existing flag and `Value` is the value currently associated with it.

Usage:

- *The following properties should hold upon exit:*
  `FlagName` is an atom. (basic_props:atm/1)
  `Value` is any term. (basic_props:term/1)

prolog_flag/3: PREDICATE

`prolog_flag(FlagName,OldValue,NewValue)`

`FlagName` is an existing flag, unify `OldValue` with the value associated with it, and set it to new value `NewValue`.

Usage 1: `prolog_flag(A,B,C)`

– *The following properties should hold at call time:*

`C` is any term. (basic_props:term/1)

`C` is currently a term which is not a free variable. (term_typing:nonvar/1)

– *The following properties should hold upon exit:*

`A` is an atom. (basic_props:atm/1)

`B` is any term. (basic_props:term/1)

Usage 2: `prolog_flag(FlagName,OldValue,NewValue)`

– *Description:* Same as `current_prolog_flag(FlagName, OldValue)`. `OldValue` and `NewValue` must be strictly identical variables.

– *The following properties should hold at call time:*

`OldValue` is a free variable. (term_typing:var/1)

`NewValue` is a free variable. (term_typing:var/1)

`FlagName` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*

`OldValue` is any term. (basic_props:term/1)

`NewValue` is any term. (basic_props:term/1)

push_prolog_flag/2: PREDICATE

`push_prolog_flag(Flag,NewValue)`

Same as `set_prolog_flag/2`, but storing current value of `Flag` to restore it with `pop_prolog_flag/1`.

Usage:

– *The following properties should hold at call time:*

`Flag` is an atom. (basic_props:atm/1)

`NewValue` is any term. (basic_props:term/1)

– *The following properties should hold upon exit:*

`Flag` is an atom. (basic_props:atm/1)

`NewValue` is any term. (basic_props:term/1)

pop_prolog_flag/1: PREDICATE

`pop_prolog_flag(Flag)`

Restore the value of `Flag` previous to the last non-canceled `push_prolog_flag/2` on it.

Usage:

– *The following properties should hold at call time:*

`Flag` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*

`Flag` is an atom. (basic_props:atm/1)

set_ciao_flag/2: PREDICATE

Usage: `set_ciao_flag(FlagName,Value)`

– *The following properties should hold globally:*

`set_ciao_flag(FlagName,Value)` is equivalent to `set_prolog_flag(FlagName,Value)`. (basic_props:equiv/2)

current_ciao_flag/2: PREDICATE

Usage: `current_ciao_flag(FlagName,Value)`

– *The following properties should hold globally:*

`current_ciao_flag(FlagName,Value)` is equivalent to `current_prolog_flag(FlagName,Value)`. (basic_props:equiv/2)

ciao_flag/3: PREDICATE

Usage: `ciao_flag(Flag,Old,New)`

– *The following properties should hold globally:*

`ciao_flag(Flag,Old,New)` is equivalent to `prolog_flag(Flag,Old,New)`. (basic_props:equiv/2)

push_ciao_flag/2: PREDICATE

Usage: `push_ciao_flag(Flag,NewValue)`

– *The following properties should hold globally:*

`push_ciao_flag(Flag,NewValue)` is equivalent to `push_prolog_flag(Flag,NewValue)`. (basic_props:equiv/2)

pop_ciao_flag/1: PREDICATE

Usage: `pop_ciao_flag(Flag)`

– *The following properties should hold globally:*

`pop_ciao_flag(Flag)` is equivalent to `pop_prolog_flag(Flag)`. (basic_props:equiv/2)

prompt/2: PREDICATE

`prompt(Old,New)`

Unify `Old` with the current prompt for reading, change it to `New`.

Usage 1: `prompt(A,B)`

– *The following properties should hold at call time:*

`B` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*

`A` is an atom. (basic_props:atm/1)

Usage 2: `prompt(Old,New)`

– *Description:* Unify `Old` with the current prompt for reading without changing it. On calls, `Old` and `New` must be strictly identical variables.
– *The following properties should hold at call time:*
`Old` is a free variable. (term_typing:var/1)
`New` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Old` is an atom. (basic_props:atm/1)
`New` is an atom. (basic_props:atm/1)

gc/0: PREDICATE

Usage:

– *Description:* Enable garbage collection. Equivalent to `set_prolog_flag(gc, on)`
– *The following properties should hold globally:*
`gc` is equivalent to `set_prolog_flag(gc,on)`. (basic_props:equiv/2)

nogc/0: PREDICATE

Usage:

– *Description:* Disable garbage collection. Equivalent to `set_prolog_flag(gc, off)`
– *The following properties should hold globally:*
`nogc` is equivalent to `set_prolog_flag(gc,off)`. (basic_props:equiv/2)

fileerrors/0: PREDICATE

Usage:

– *Description:* Enable reporting of file errors. Equivalent to `set_prolog_flag(fileerrors, on)`
– *The following properties should hold globally:*
`fileerrors` is equivalent to `set_prolog_flag(fileerrors,on)`. (basic_props:equiv/2)

nofileerrors/0: PREDICATE

Usage:

– *Description:* Disable reporting of file errors. Equivalent to `set_prolog_flag(fileerrors, off)`
– *The following properties should hold globally:*
`nofileerrors` is equivalent to `set_prolog_flag(fileerrors,off)`. (basic_props:equiv/2)

24.3 Documentation on multifiles (prolog_flags)

define_flag/3: PREDICATE

`define_flag(Flag,Values,Default)`

New flags can be defined by writing facts of this predicate. `Flag` is the name of the new flag, `Values` defines the posible values for the flag (see below) and `Default` defines the predefined value associated with the flag (which should be compatible with `Values`).

The predicate is *multifile*.

Usage: `define_flag(Flag,FlagValues,Default)`

– *The following properties hold upon exit:*

`Flag` is an atom. (basic_props:atm/1)

Define the valid flag values (basic_props:flag_values/1)

24.4 Documentation on internals (prolog_flags)

set_prolog_flag/1: PREDICATE

No further documentation available for this predicate.

24.5 Known bugs and planned improvements (prolog_flags)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

25 Fast/concurrent update of facts

Author(s): Daniel Cabeza, Manuel Carro.

Prolog implementations traditionally implement the concept of dynamic predicates: predicates which can be inspected or modified at run-time, adding or deleting individual clauses. The power of this feature comes at a cost: as new clause bodies can be arbitrarily added to the program, new predicate calls can arise which are not 'visible' at compile-time, thus complicating global analysis and optimization of the code. But it is the case that most of the time what the programmer wants is simply to store data, with the purpose of sharing it between search branches, predicates, or even execution threads. In Ciao the concept of data predicate serves this purpose: a data predicate is a predicate composed exclusively by facts, which can be inspected, and dynamically added or deleted, at run-time. Using data predicates instead of normal dynamic predicates brings benefits in terms of speed, but above all makes the code much easier to analyze automatically and thus allows better optimization.

Also, a special kind of data predicates exists, *concurrent predicates*, which can be used to communicate/synchronize among different execution threads (see Chapter 88 [Low-level concurrency/multithreading primitives], page 465).

Data predicates must be declared through a `data/1` declaration. Concurrent data predicates must be declared through a `concurrent/1` declaration.

25.1 Usage and interface (`data_facts`)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `asserta_fact/1`, `asserta_fact/2`, `assertz_fact/1`, `assertz_fact/2`, `current_fact/1`, `current_fact/2`, `retract_fact/1`, `retractall_fact/1`, `current_fact_nb/1`, `retract_fact_nb/1`, `close_predicate/1`, `open_predicate/1`, `set_fact/1`, `erase/1`.
  - *Regular Types:*
    `reference/1`.

25.2 Documentation on exports (`data_facts`)

asserta_fact/1: PREDICATE

`asserta_fact(Fact)`

`Fact` is added to the corresponding data predicate. The fact becomes the first clause of the predicate concerned.

Meta-predicate with arguments: `asserta_fact(fact)`.

Usage:

- *The following properties should hold at call time:*
  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

asserta_fact/2: PREDICATE

`asserta_fact(Fact,Ref)`

Same as `asserta_fact/1`, instantiating `Ref` to a unique identifier of the asserted fact.

Meta-predicate with arguments: `asserta_fact(fact,?)`.

Usage:

– *The following properties should hold at call time:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Ref` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Ref` is a reference of a dynamic or data clause. (data_facts:reference/1)

assertz_fact/1: PREDICATE

`assertz_fact(Fact)`

`Fact` is added to the corresponding data predicate. The fact becomes the last clause of the predicate concerned.

Meta-predicate with arguments: `assertz_fact(fact)`.

Usage:

– *The following properties should hold at call time:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

assertz_fact/2: PREDICATE

`assertz_fact(Fact,Ref)`

Same as `assertz_fact/1`, instantiating `Ref` to a unique identifier of the asserted fact.

Meta-predicate with arguments: `assertz_fact(fact,?)`.

Usage:

– *The following properties should hold at call time:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Ref` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Ref` is a reference of a dynamic or data clause. (data_facts:reference/1)

current_fact/1: PREDICATE

`current_fact(Fact)`

Gives on backtracking all the facts defined as data or concurrent which unify with `Fact`. It is faster than calling the predicate explicitly, which do invoke the meta-interpreter. If the `Fact` has been defined as concurrent and has not been closed, `current_fact/1` will wait (instead of failing) for more clauses to appear after the last clause of `Fact` is returned.

Meta-predicate with arguments: `current_fact(fact)`.

Usage:

– *The following properties should hold at call time:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

current_fact/2: PREDICATE

`current_fact(Fact,Ref)`

`Fact` is a fact of a data predicate and `Ref` is its reference identifying it uniquely.

Meta-predicate with arguments: `current_fact(fact,?)`.

Usage 1:

– *Description:* Gives on backtracking all the facts defined as data which unify with `Fact`, instantiating `Ref` to a unique identifier for each fact.

– *The following properties should hold at call time:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Ref` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Ref` is a reference of a dynamic or data clause. (data_facts:reference/1)

Usage 2:

– *Description:* Given `Ref`, unifies `Fact` with the fact identified by it.

– *Call and exit should be compatible with:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold at call time:*

`Ref` is a reference of a dynamic or data clause. (data_facts:reference/1)

– *The following properties should hold upon exit:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

retract_fact/1: PREDICATE

`retract_fact(Fact)`

Unifies `Fact` with the first matching fact of a data predicate, and then erases it. On backtracking successively unifies with and erases new matching facts. If `Fact` is declared as concurrent and is non- closed, `retract_fact/1` will wait for more clauses or for the closing of the predicate after the last matching clause has been removed.

Meta-predicate with arguments: `retract_fact(fact)`.

Usage:

– *The following properties should hold at call time:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

retractall_fact/1: PREDICATE

`retractall_fact(Fact)`

Erase all the facts of a data predicate unifying with `Fact`. Even if all facts are removed, the predicate continues to exist.

Meta-predicate with arguments: `retractall_fact(fact)`.

Usage:

- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

- *The following properties should hold upon exit:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

current_fact_nb/1: PREDICATE

`current_fact_nb(Fact)`

Behaves as `current_fact/1` but a fact is never waited on even if it is concurrent and non-closed.

Meta-predicate with arguments: `current_fact_nb(fact)`.

Usage:

- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

- *The following properties should hold upon exit:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

retract_fact_nb/1: PREDICATE

`retract_fact_nb(Fact)`

Behaves as `retract_fact/1`, but never waits on a fact, even if it has been declared as concurrent and is non- closed.

Meta-predicate with arguments: `retract_fact_nb(fact)`.

Usage:

- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

- *The following properties should hold upon exit:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

close_predicate/1: PREDICATE

`close_predicate(Pred)`

Changes the behavior of the predicate `Pred` if it has been declared as a concurrent predicate: calls to this predicate will fail (instead of wait) if no more clauses of `Pred` are available.

Meta-predicate with arguments: `close_predicate(fact)`.

Usage:

– *The following properties should hold at call time:*

`Pred` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Pred` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

open_predicate/1: PREDICATE

`open_predicate(Pred)`

Reverts the behavior of concurrent predicate `Pred` to waiting instead of failing if no more clauses of `Pred` are available.

Meta-predicate with arguments: `open_predicate(fact)`.

Usage:

– *The following properties should hold at call time:*

`Pred` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Pred` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

set_fact/1: PREDICATE

`set_fact(Fact)`

Sets `Fact` as the unique fact of the corresponding data predicate.

Meta-predicate with arguments: `set_fact(fact)`.

Usage:

– *The following properties should hold at call time:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

erase/1: PREDICATE

`erase(Ref)`

Deletes the clause referenced by `Ref`.

Usage:

– *The following properties should hold at call time:*

`Ref` is a reference of a dynamic or data clause. (data_facts:reference/1)

– *The following properties should hold upon exit:*

`Ref` is a reference of a dynamic or data clause. (data_facts:reference/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

reference/1: REGTYPE

Usage: `reference(R)`

– *Description:* `R` is a reference of a dynamic or data clause.

25.3 Documentation on internals (`data_facts`)

data/1: DECLARATION

Usage: `:- data Predicates.`

– *Description:* Defines each predicate in `Predicates` as a data predicate. If a predicate is defined data in a file, it must be defined data in every file containing clauses for that predicate. The directive should precede all clauses of the affected predicates. This directive is defined as a prefix operator in the compiler.

– *The following properties hold at call time:*

`Predicates` is a sequence or list of `prednames`. (basic_props:sequence_or_list/2)

concurrent/1: DECLARATION

Usage: `:- concurrent Predicates.`

– *Description:* Defines each predicate in `Predicates` as a concurrent predicate. If a predicate is defined concurrent in a file, it must be defined concurrent in every file containing clauses for that predicate. The directive should precede all clauses of the affected predicates. This directive is defined as a prefix operator in the compiler.

– *The following properties hold at call time:*

`Predicates` is a sequence or list of `prednames`. (basic_props:sequence_or_list/2)

25.4 Known bugs and planned improvements (`data_facts`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

26 Extending the syntax

Author(s): Daniel Cabeza.

This chapter documents the builtin directives in Ciao for extending the syntax of source files.

Note that the ISO-Prolog directive `char_conversion/2` is not implemented, since Ciao does not (yet) have a character conversion table.

26.1 Usage and interface (`syntax_extensions`)

- **Library usage:**
  These directives are builtin in Ciao, so nothing special has to be done to use them.

26.2 Documentation on internals (`syntax_extensions`)

op/3: DECLARATION

Usage: :- `op(Priority,Op_spec,Operator)`. ⟨• ISO •⟩

– *Description:* Updates the operator table for reading the terms in the rest of the current text, in the same way as the builtin `op/3` does. Its scope is local to the current text. Usually included in package files.

– *The following properties hold at call time:*

`Priority` is an integer. (basic_props:int/1)

`Op_spec` specifies the type and associativity of an operator. (basic_props:operator_specifier/1)

`Operator` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

new_declaration/1: DECLARATION

Usage: :- `new_declaration(Predicate)`.

– *Description:* Declares `Predicate` to be a valid declaration in the rest of the current text. Such declarations are simply ignored by the compiler or top level, but can be used by other code processing programs such as an automatic documentator. Also, they can easily translated into standard code (a set of facts and/or rules) by defining a suitable expansion (e.g., by `add_sentence_trans/1`, etc.). This is tipically done in package files.

Equivalent to `new_declaration(Predicate, off)`.

– *The following properties hold at call time:*

`Predicate` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

new_declaration/2: DECLARATION

Usage: :- `new_declaration(Predicate,In_Itf)`.

– *Description:* Declares `Predicate` to be a valid declaration in the rest of the current text. Such declarations will be included in the interface file for this file if `In_Itf` is 'on', not if it is 'off'. Including such declarations in interface files makes them visible while processing other modules which make use of this one.

– *The following properties hold at call time:*

`Predicate` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

`In_Itf` is 'on' or 'off' (syntax_extensions:switch/1)

load_compilation_module/1: DECLARATION

Usage: :- `load_compilation_module(File)`.

– *Description:* Loads code defined in `File` into the compiler, usually including predicates which define translations of clauses, sentences, and terms, for use with the declarations `add_sentence_trans/2` and similar ones. The application order of translations is determined by ascending *priority* numbers. Normally included in package files.

– *The following properties hold at call time:*

`File` is a source name. (streams_basic:sourcename/1)

add_sentence_trans/2: DECLARATION

Usage: :- `add_sentence_trans(Predicate,Priority)`.

– *Description:* Starts a translation, defined by `Predicate`, of the terms read by the compiler in the rest of the current text. For each subsequent term read by the compiler, the translation predicate is called to obtain a new term which will be used by the compiler as if it where the term present in the file. If the call fails, the term is used as such. A list may be returned also, to translate a single term into several terms. Before calling the translation predicate with actual program terms, it is called with an input of `0` to give an opportunity of making initializations for the module, discarding the result (note that normally a 0 could not be there). `Predicate` must be exported by a module previously loaded with a `load_compilation_module/1` declaration. Normally included in package files.

– *The following properties hold at call time:*

`Predicate` is a translation predicate spec (has arity 2 or 3). (syntax_extensions:translation_predname/1)

`Priority` is an integer. (basic_props:int/1)

add_term_trans/2: DECLARATION

Usage: :- `add_term_trans(P,Priority)`.

– *Description:* Starts a translation, defined by `Predicate`, of the terms and sub-terms read by the compiler in the rest of the current text. This translation is performed after all translations defined by `add_sentence_trans/1` are done. For each subsequent term read by the compiler, and recursively any subterm included, the translation predicate is called to possibly obtain a new term to replace the old one. Care must be taken of not introducing an endless loop of translations. `Predicate` must be exported by a module previously loaded with a `load_compilation_module/1` declaration. Normally included in package files.

– *The following properties hold at call time:*

`P` is a translation predicate spec (has arity 2 or 3). (syntax_extensions:translation_predname/1)

`Priority` is an integer. (basic_props:int/1)

add_goal_trans/2: DECLARATION

Usage: :- `add_goal_trans(Predicate,Priority)`.

– *Description:* Declares a translation, defined by `Predicate`, of the goals present in the clauses of the current text. This translation is performed after all translations defined by `add_sentence_trans/1` and `add_term_trans/1` are done. For each clause read by the compiler, the translation predicate is called with each goal present in the clause to possibly obtain other goal to substitute the original one, and the translation is subsequently applied to the resulting goal. Care must be taken of not introducing an endless loop of translations. `Predicate` must be exported by a module previously loaded with a `load_compilation_module/1` declaration. Bear in mind that this type of translation noticeably slows down compilation. Normally included in package files.

– *The following properties hold at call time:*

`Predicate` is a translation predicate spec (has arity 2 or 3). (syntax_extensions:translation_predname/1)

`Priority` is an integer. (basic_props:int/1)

add_clause_trans/2: DECLARATION

Usage: :- `add_clause_trans(Predicate,Priority)`.

– *Description:* Declares a translation, defined by `Predicate`, of the clauses of the current text. The translation is performed before `add_goal_trans/1` translations but after `add_sentence_trans/1` and `add_term_trans/1` translations. The usefulness of this translation is that information of the interface of related modules is available when it is performed. For each clause read by the compiler, the translation predicate is called with the first argument instantiated to a structure `clause(Head,Body)`, and the predicate must return in the second argument a similar structure, without changing the functor in `Head` (or fail, in which case the clause is used as is). Before executing the translation predicate with actual clauses it is called with an input of `clause(0,0)`, discarding the result.

– *The following properties hold at call time:*

`Predicate` is a translation predicate spec (has arity 2 or 3). (syntax_extensions:translation_predname/1)

`Priority` is an integer. (basic_props:int/1)

translation_predname/1: REGTYPE

A translation predicate is a predicate of arity 2 or 3 used to make compile-time translations. The compiler invokes a translation predicate instantiating its first argument with the item to be translated, and if the predicate is of arity 3 its third argument with the name of the module where the translation is done. If the call is successful, the second argument is used as if that item were in the place of the original, else the original item is used.

Usage: `translation_predname(P)`

– *Description:* `P` is a translation predicate spec (has arity 2 or 3).

27 Message printing primitives

Author(s): Daniel Cabeza, Edison Mera (improvements).

This module provides predicates for printing in a unified way informational messages, and also for printing some terms in a specific way.

27.1 Usage and interface (io_aux)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `message/2`, `message_lns/4`, `messages/1`, `error/1`, `warning/1`, `note/1`, `message/1`, `debug/1`, `inform_user/1`, `display_string/1`, `display_list/1`, `display_term/1`, `add_lines/4`.
  - *Regular Types:*
    `message_info/1`, `message_type/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

27.2 Documentation on exports (io_aux)

message/2: PREDICATE

Usage: `message(Type,Message)`

- *Description:* Output to standard error `Message`, which is of type `Type`. The `quiet` *prolog flag* (see Chapter 24 [Changing system behaviour and various flags], page 181) controls which messages are actually output, depending on its type. Also, for `error`, `warning` and `note` messages, a prefix is output which denotes the severity of the message.
- *The following properties should hold at call time:*
  Specifies the different types of messages. (io_aux:message_type/1)
  `Message` is an item or a list of items from this list:

  `$$(String)` `String` is a string, which is output with `display_string/1`.

  `''(Term)` `Term` is output quoted. If the module `write` is loaded, the term is output with `writeq/1`, else with `displayq/1`.

  `~~(Term)` `Term` is output unquoted. If the module `write` is loaded, the term is output with `write/1`, else with `display/1`.

  `[](Term)` `Term` is recursively output as a message, can be an item or a list of items from this list.

  `Term` Any other term is output with `display/1`.

  (io_aux:message_text/1)

message_lns/4: PREDICATE

Usage: `message_lns(Type,L0,L1,Message)`

– *Description:* Output to standard error `Message`, which is of type `Type`, and occurs between lines `L0` and `L1`. This is the same as `message/2`, but printing the lines where the message occurs in a unified way (this is useful because automatic tools such as the emacs mode know how to parse them).

– *The following properties should hold at call time:*

Specifies the different types of messages. (io_aux:message_type/1)

`L0` is a non-negative integer. (basic_props:nnegint/1)

`L1` is a non-negative integer. (basic_props:nnegint/1)

`Message` is an item or a list of items from this list:

`$$(String)` `String` is a string, which is output with `display_string/1`.

`''(Term)` `Term` is output quoted. If the module `write` is loaded, the term is output with `writeq/1`, else with `displayq/1`.

`~~(Term)` `Term` is output unquoted. If the module `write` is loaded, the term is output with `write/1`, else with `display/1`.

`[](Term)` `Term` is recursively output as a message, can be an item or a list of items from this list.

`Term` Any other term is output with `display/1`.

(io_aux:message_text/1)

messages/1: PREDICATE

Usage: `messages(Messages)`

– *Description:* Print each element in `Messages` using `message/2`, `message_lns/4`, `message/1`, `error/1`, `warning/1`, `note/1` or `debug/1` predicate. If the element should be printed using message_lns/4, it is printed in a compact way, avoiding to print the same file name several times.

– *The following properties should hold at call time:*

`Messages` is a list of `message_info`s. (basic_props:list/2)

error/1: PREDICATE

Defined as

```
error(Message) :-
        message(error,Message).
```

.

warning/1: PREDICATE

Defined as

```
warning(Message) :-
        message(warning,Message).
```

.

note/1: PREDICATE

Defined as

```
note(Message) :-
        message(note,Message).
```

.

message/1: PREDICATE

Defined as

```
message(Message) :-
        message(message,Message).
```

.

debug/1: PREDICATE

Defined as

```
debug(Message) :-
        message(debug,Message).
```

.

inform_user/1: PREDICATE

`inform_user(Message)`

Similar to `message/1`, but `Message` is output with `display_list/1`. This predicate is obsolete, and may disappear in future versions.

display_string/1: PREDICATE

`display_string(String)`

Output `String` as the sequence of characters it represents.

Usage: `display_string(String)`

– *The following properties should hold at call time:*

`String` is a string (a list of character codes). (basic_props:string/1)

display_list/1: PREDICATE

`display_list(List)`

Outputs `List`. If `List` is a list, do `display/1` on each of its elements, else do `display/1` on `List`.

display_term/1: PREDICATE

`display_term(Term)`

Output `Term` in a way that a `read/1` will be able to read it back, even if operators change.

message_info/1: REGTYPE

Usage:

– *Description:* The type of the elements to be printed using the `messages/1` predicate. Defined as

```
message_info(message_lns(Source,Ln0,Ln1,Type,Text)) :-
        atm(Source),
        nnegint(Ln0),
        nnegint(Ln1),
        message_type(Type),
        message_text(Text).
message_info(message(Type,Text)) :-
        atm(Type),
        message_text(Text).
message_info(error(Text)) :-
        message_text(Text).
message_info(warning(Text)) :-
        message_text(Text).
message_info(note(Text)) :-
        message_text(Text).
message_info(message(Text)) :-
        message_text(Text).
message_info(debug(Text)) :-
        message_text(Text).
```

.

message_type/1: REGTYPE

Usage:

– *Description:* Specifies the different types of messages.

add_lines/4: PREDICATE

No further documentation available for this predicate.

27.3 Known bugs and planned improvements (`io_aux`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.
- `message/2` assumes that a module with name 'write' is library(write).

28 Attributed variables

Author(s): Christian Holzbaur, Daniel Cabeza, Manuel Carro.

These predicates allow the manipulation of *attributed variables*. Attributes are special terms which are attached to a (free) variable, and are hidden from the normal Prolog computation. They can only be treated by using the predicates below.

28.1 Usage and interface (attributes)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `attach_attribute/2`, `get_attribute/2`, `update_attribute/2`, `detach_attribute/1`.
  - *Multifiles:*
    `verify_attribute/2`, `combine_attributes/2`.

28.2 Documentation on exports (attributes)

attach_attribute/2: PREDICATE

Usage: `attach_attribute(Var,Attr)`

- *Description:* Attach attribute `Attr` to `Var`.
- *The following properties should hold at call time:*
  `Var` is a free variable. (term_typing:var/1)
  `Attr` is currently a term which is not a free variable. (term_typing:nonvar/1)
- *The following properties hold upon exit:*
  `Var` is a free variable. (term_typing:var/1)
  `Attr` is currently a term which is not a free variable. (term_typing:nonvar/1)

get_attribute/2: PREDICATE

Usage: `get_attribute(Var,Attr)`

- *Description:* Unify `Attr` with the attribute of `Var`, or fail if `Var` has no attribute.
- *The following properties should hold at call time:*
  `Var` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*
  `Attr` is currently a term which is not a free variable. (term_typing:nonvar/1)

update_attribute/2: PREDICATE

Usage: `update_attribute(Var,Attr)`

- *Description:* Change the attribute of attributed variable `Var` to `Attr`.

– *The following properties should hold at call time:*
`Var` is a free variable. (term_typing:var/1)
`Attr` is currently a term which is not a free variable. (term_typing:nonvar/1)
– *The following properties hold upon exit:*
`Var` is a free variable. (term_typing:var/1)
`Attr` is currently a term which is not a free variable. (term_typing:nonvar/1)

detach_attribute/1: PREDICATE

Usage: `detach_attribute(Var)`

– *Description:* Take out the attribute from the attributed variable `Var`.
– *The following properties should hold at call time:*
`Var` is a free variable. (term_typing:var/1)
– *The following properties hold upon exit:*
`Var` is a free variable. (term_typing:var/1)

28.3 Documentation on multifiles (`attributes`)

verify_attribute/2: PREDICATE

The predicate is *multifile*.

Usage: `verify_attribute(Attr,Term)`

– *Description: A user defined predicate.* This predicate is called when an attributed variable with attribute `Attr` is about to be unified with the non-variable term `Term`. The user should define this predicate (as multifile) in the modules implementing special unification.
– *The following properties should hold at call time:*
`Attr` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)
– *The following properties hold upon exit:*
`Attr` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

combine_attributes/2: PREDICATE

The predicate is *multifile*.

Usage: `combine_attributes(Var1,Var2)`

– *Description: A user defined predicate.* This predicate is called when two attributed variables with attributes `Var1` and `Var2` are about to be unified. The user should define this predicate (as multifile) in the modules implementing special unification.
– *The following properties should hold at call time:*
`Var1` is a free variable. (term_typing:var/1)
`Var2` is a free variable. (term_typing:var/1)
– *The following properties hold upon exit:*
`Var1` is a free variable. (term_typing:var/1)
`Var2` is a free variable. (term_typing:var/1)

28.4 Other information (attributes)

Note that `combine_attributes/2` and `verify_attribute/2` are not called with the attributed variables involved, but with the corresponding attributes instead. The reasons are:

- There are simple applications which only refer to the attributes.
- If the application wants to refer to the attributed variables themselves, they can be made part the attribute term. The implementation of `freeze/2` utilizes this technique. Note that this does not lead to cyclic structures, as the connection between an attributed variable and its attribute is invisible to the pure parts of the Prolog implementation.
- If attributed variables were passed as arguments, the user code would have to refer to the attributes through an extra call to `get_attribute/2`.
- As the/one attribute is the first argument to each of the two predicates, indexing applies. Note that attributed variables themselves look like variables to the indexing mechanism.

However, future improvements may change or extend the interface to attributed variables in order to provide a richer and more expressive interface.

For customized output of attributed variables, please refer to the documentation of the predicate `portray_attribute/2`.

28.5 Known bugs and planned improvements (attributes)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

29 Internal Runtime Information

Author(s): Daniel Cabeza, Manuel Carro, Jose F. Morales.

This module provides internal information about the current running engine and enviroment. That information includes the architecture, platform, operating system, location of libraries, and C header files. That information is mainly used in parts of the Ciao dynamic compilation (location of source, generation of gluecode for the foreign interface, etc.).

29.1 Usage and interface (system_info)

- **Library usage:**
  These predicates are builtin in Ciao, so nothing special has to be done to use them.
- **Exports:**
  - *Predicates:*
    `get_arch/1`, `get_os/1`, `get_platform/1`, `get_debug/1`, `get_eng_location/1`, `get_ciao_ext/1`, `get_exec_ext/1`, `get_so_ext/1`, `this_module/1`, `current_module/1`, `ciao_c_headers_dir/1`, `ciao_lib_dir/1`.
  - *Regular Types:*
    `internal_module_id/1`.

29.2 Documentation on exports (system_info)

get_arch/1: PREDICATE

This predicate will describe the computer architecture wich is currently executing the predicate.

Computer architectures are identified by a simple atom. This atom is implementation-defined, and may suffer any change from one Ciao version to another.

For example, Ciao running on an Intel-based machine will retrieve:

```
?- get_arch(I).

I = i86 ? ;

no
?-
```

Usage: `get_arch(ArchDescriptor)`

- *Description:* Unifies `ArchDescriptor` with a simple atom which describes the computer architecture currently executing the predicate.
- *The following properties hold upon exit:*
  `ArchDescriptor` is an atom. (basic_props:atm/1)

get_os/1: PREDICATE

This predicate will describe the operating system which is running on the machine currently executing the Prolog program.

Operating systems are identified by a simple atom. This atom is implementation-defined, and may suffer changes from one Ciao version to another.

For example, Ciao running on Linux will retrieve:

```
?- get_os(I).

I = 'LINUX' ? ;

no
?-
```

Usage: `get_os(OsDescriptor)`

- *Description:* Unifies `OsDescriptor` with a simple atom which describes the running operating system when predicate was called.
- *The following properties hold upon exit:*

  `OsDescriptor` is an atom. (basic_props:atm/1)

get_platform/1: PREDICATE

Usage: `get_platform(Platform)`

- *Description:* `Platform` is the atom describing the current operating system and computer architecture.
- *The following properties hold upon exit:*

  `Platform` is an atom. (basic_props:atm/1)

get_debug/1: PREDICATE

Usage: `get_debug(Debug)`

- *Description:* Unifies `Debug` with an atom that indicates if the emulator has been compiled with debug information
- *The following properties hold upon exit:*

  `Debug` is an atom. (basic_props:atm/1)

get_eng_location/1: PREDICATE

Usage: `get_eng_location(Ext)`

- *Description:* `Ext` indicates if the engine is located in a library (`dyn`) or in an executable (`empty`).
- *Calls should, and exit will be compatible with:*

  `Ext` is an atom. (basic_props:atm/1)

get_ciao_ext/1: PREDICATE

Usage: `get_ciao_ext(Ext)`

- *Description:* `Ext` is the default extension for the executable Ciao programs.
- *The following properties hold upon exit:*

  `Ext` is an atom. (basic_props:atm/1)

get_exec_ext/1: PREDICATE

Usage: `get_exec_ext(Ext)`

- *Description:* `Ext` is the extension for executables.
- *The following properties hold upon exit:*

  `Ext` is an atom. (basic_props:atm/1)

get_so_ext/1: PREDICATE

Usage: `get_so_ext(Ext)`

- *Description:* `Ext` is the default extension for the shared libraries. For example, `.dll` in Windows and `.so` in most Unix systems.
- *Calls should, and exit will be compatible with:*

  `Ext` is an atom. (basic_props:atm/1)

this_module/1: PREDICATE

Meta-predicate with arguments: `this_module(addmodule(?))`.

Usage: `this_module(Module)`

- *Description:* `Module` is the internal module identifier for current module.
- *The following properties hold upon exit:*

  `Module` is an internal module identifier (system_info:internal_module_id/1)

current_module/1: PREDICATE

This predicate will successively unify its argument with all module names currently loaded. Module names will be simple atoms.

When called using a free variable as argument, it will retrieve on backtracking all modules currently loaded. This is usefull when called from the Ciao `toplevel`.

When called using a module name as argument it will check whether the given module is loaded or not. This is usefull when called from user programs.

Usage: `current_module(Module)`

- *Description:* Retrieves (on backtracking) all currently loaded modules into your application.
- *The following properties should hold upon exit:*

  `Module` is an internal module identifier (system_info:internal_module_id/1)
- *The following properties should hold globally:*

  This predicate is understood natively by CiaoPP. (basic_props:native/1)

ciao_c_headers_dir/1: PREDICATE

Usage: `ciao_c_headers_dir(CiaoPath)`

- *Description:* `CiaoPath` is the path to the root of the installed Ciao header C files (.h), typically used for interfacing Ciao and C.
- *The following properties hold upon exit:*

  `CiaoPath` is an atom. (basic_props:atm/1)

ciao_lib_dir/1: PREDICATE

Usage: `ciao_lib_dir(CiaoPath)`

– *Description:* `CiaoPath` is the path to the root of the Ciao libraries. Inside this directory, there are the directories 'lib', 'library' and 'contrib', which contain library modules.

– *The following properties hold upon exit:*

`CiaoPath` is an atom. (basic_props:atm/1)

internal_module_id/1: REGTYPE

For a user file it is a term user/1 with an argument different for each user file, for other modules is just the name of the module (as an atom).

Usage: `internal_module_id(M)`

– *Description:* `M` is an internal module identifier

29.3 Known bugs and planned improvements (`system_info`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

30 Conditional Compilation

Author(s): Jose F. Morales.

This package defines a serie of directives for conditional compilation that allow the inclusion or exclusion of code blocks (which may contain nested conditional directives) based on the truth value at compile time of special goals called *conditions*. The syntax for conditional directives is:

```
:- if(Cond1).
  <<Block1>>
:- elif(Cond2).
  <<Block2>>
:- else.
  <<BlockN>>
:- endif.
```

where `elif(_)` can appear zero or more times and the `else` part is optional. The valid conditions are:

- calls to `current_prolog_flag/2`.
- conjunctions, disjunctions, or negations of conditions.

The sentences in `Block1` are included if the condition in `Cond1` is satisfied, else `Block2` is included if `Cond2` is satisfied (and so on for each `elif`), and `BlockN` if no previous condition is satisfied.

30.1 Usage and interface (`condcomp_doc`)

- **Library usage:**
  The conditional compilation directives are enabled by including the `condcomp` package in the package list of a module or by means of an explicit `:- use_package(condcomp)`

30.2 Known bugs and planned improvements (`condcomp_doc`)

- This package implements a `:- define(X)` directive and `defined(X)` condition. However, the syntax and semantic of this feature has still to be decided. Do not use it.
- Errors do not show line numbers

31 Other predicates and features defined by default

Author(s): Daniel Cabeza.

To simplify the use of Ciao Prolog to the first-timers, some other predicates and features are defined by default in normal cases, to provide more or less what other prologs define by default. Here are explicitly listed the predicates defined, coming from several libraries. Apart from those, the features defined in Chapter 41 [Definite clause grammars], page 271 and Chapter 53 [Enabling operators at run-time], page 335 are also activated.

31.1 Usage and interface (`default_predicates`)

- **Library usage:**
  No need of explicit loading. It is included by default in modules starting with a `module/2` declaration or user files without a starting `use_package/1` declaration. In the Ciao shell, it is loaded by default when no `~/.ciaorc` exists. Note that `:- module(`*modulename*`,`*exports*`)` is equivalent to `:- module(`*modulename*`,`*exports*`,[default])` If you do not want these predicates/features loaded for a given file (in order to make the executable smaller) you can ask for this explicitly using `:- module(`*modulename*`,`*exports*`,[])` or in a user file `:- use_package([])`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `read`, `write`, `operators`, `iso_byte_char`, `iso_misc`, `format`, `lists`, `sort`, `between`, `compiler/compiler`, `system`, `prolog_sys`, `dec10_io`, `old_database`, `ttyout`.

31.2 Documentation on exports (`default_predicates`)

op/3: (UNDOC_REEXPORT)
Imported from `operators` (see the corresponding documentation for details).

current_op/3: (UNDOC_REEXPORT)
Imported from `operators` (see the corresponding documentation for details).

append/3: (UNDOC_REEXPORT)
Imported from `lists` (see the corresponding documentation for details).

delete/3: (UNDOC_REEXPORT)
Imported from `lists` (see the corresponding documentation for details).

select/3: (UNDOC_REEXPORT)
Imported from `lists` (see the corresponding documentation for details).

nth/3: (UNDOC_REEXPORT)
Imported from `lists` (see the corresponding documentation for details).

last/2: (UNDOC_REEXPORT)
Imported from `lists` (see the corresponding documentation for details).

reverse/2: (UNDOC_REEXPORT)
Imported from `lists` (see the corresponding documentation for details).

length/2: (UNDOC_REEXPORT)
Imported from `lists` (see the corresponding documentation for details).

use_module/1: (UNDOC_REEXPORT)
Imported from `compiler` (see the corresponding documentation for details).

use_module/2: (UNDOC_REEXPORT)
Imported from `compiler` (see the corresponding documentation for details).

ensure_loaded/1: (UNDOC_REEXPORT)
Imported from `compiler` (see the corresponding documentation for details).

^/2: (UNDOC_REEXPORT)
Imported from `aggregates` (see the corresponding documentation for details).

findnsols/5: (UNDOC_REEXPORT)
Imported from `aggregates` (see the corresponding documentation for details).

findnsols/4: (UNDOC_REEXPORT)
Imported from `aggregates` (see the corresponding documentation for details).

findall/4: (UNDOC_REEXPORT)
Imported from `aggregates` (see the corresponding documentation for details).

findall/3: (UNDOC_REEXPORT)
Imported from `aggregates` (see the corresponding documentation for details).

bagof/3: (UNDOC_REEXPORT)
Imported from `aggregates` (see the corresponding documentation for details).

setof/3: (UNDOC_REEXPORT)
Imported from `aggregates` (see the corresponding documentation for details).

wellformed_body/3: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

data/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

dynamic/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

current_predicate/2: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

current_predicate/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

clause/3: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

clause/2: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

abolish/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

retractall/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

retract/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

assert/2: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

assert/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

assertz/2: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

assertz/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

asserta/2: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

asserta/1: (UNDOC_REEXPORT)
Imported from `dynamic` (see the corresponding documentation for details).

read_option/1: (UNDOC_REEXPORT)
Imported from `read` (see the corresponding documentation for details).

second_prompt/2: (UNDOC_REEXPORT)
Imported from `read` (see the corresponding documentation for details).

read_top_level/3: (UNDOC_REEXPORT)
Imported from `read` (see the corresponding documentation for details).

read_term/3: (UNDOC_REEXPORT)
Imported from `read` (see the corresponding documentation for details).

read_term/2: (UNDOC_REEXPORT)
Imported from `read` (see the corresponding documentation for details).

read/2: (UNDOC_REEXPORT)
Imported from `read` (see the corresponding documentation for details).

read/1: (UNDOC_REEXPORT)
Imported from `read` (see the corresponding documentation for details).

write_attribute/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

printable_char/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

prettyvars/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

numbervars/3: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

portray_clause/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

portray_clause/2: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

write_list1/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

print/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

print/2: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

write_canonical/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

write_canonical/2: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

writeq/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

writeq/2: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

write/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

write/2: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

write_option/1: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

write_term/2: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

write_term/3: (UNDOC_REEXPORT)
Imported from `write` (see the corresponding documentation for details).

put_char/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

put_char/1: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

peek_char/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

peek_char/1: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

get_char/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

get_char/1: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

put_byte/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

put_byte/1: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

peek_byte/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

peek_byte/1: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

get_byte/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

get_byte/1: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

char_codes/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

number_chars/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

atom_chars/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

char_code/2: (UNDOC_REEXPORT)
Imported from `iso_byte_char` (see the corresponding documentation for details).

unify_with_occurs_check/2: (UNDOC_REEXPORT)
Imported from `iso_misc` (see the corresponding documentation for details).

sub_atom/5: (UNDOC_REEXPORT)
Imported from `iso_misc` (see the corresponding documentation for details).

compound/1: (UNDOC_REEXPORT)
Imported from `iso_misc` (see the corresponding documentation for details).

once/1: (UNDOC_REEXPORT)
Imported from `iso_misc` (see the corresponding documentation for details).

format_control/1: (UNDOC_REEXPORT)
Imported from `format` (see the corresponding documentation for details).

format_to_string/3: (UNDOC_REEXPORT)
Imported from `format` (see the corresponding documentation for details).

sformat/3: (UNDOC_REEXPORT)
Imported from `format` (see the corresponding documentation for details).

format/3: (UNDOC_REEXPORT)
Imported from `format` (see the corresponding documentation for details).

format/2: (UNDOC_REEXPORT)
Imported from `format` (see the corresponding documentation for details).

keypair/1: (UNDOC_REEXPORT)
Imported from `sort` (see the corresponding documentation for details).

keylist/1: (UNDOC_REEXPORT)
Imported from `sort` (see the corresponding documentation for details).

keysort/2: (UNDOC_REEXPORT)
Imported from `sort` (see the corresponding documentation for details).

sort/2: (UNDOC_REEXPORT)
Imported from `sort` (see the corresponding documentation for details).

between/3: (UNDOC_REEXPORT)
Imported from `between` (see the corresponding documentation for details).

system_error_report/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

replace_characters/4: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

no_swapslash/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

cyg2win/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

winpath_c/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

winpath/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

winpath/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

using_windows/0: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

rename_file/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

delete_directory/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

delete_file/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

set_exec_mode/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

chmod/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

chmod/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

fmode/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

modif_time0/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

modif_time/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_properties/6: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_property/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_exists/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_exists/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

mktemp_in_tmp/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

mktemp/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

directory_files/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

wait/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

exec/8: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

exec/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

exec/4: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

popen_mode/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

popen/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

system/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

system/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

shell/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

shell/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

shell/0: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

cd/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

working_directory/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

make_dirpath/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

make_dirpath/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

make_directory/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

make_directory/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

umask/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

current_executable/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

current_host/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_address/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_tmp_dir/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_grnam/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_pwnam/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_gid/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_uid/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_pid/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_dir_name/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

extract_paths/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

dir_path/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

copy_file/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

copy_file/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

c_errno/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

del_env/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

set_env/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

current_env/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

setenvstr/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

getenvstr/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

datime_struct/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

datime/9: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

datime/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

time/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

pause/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

garbage_collect/0: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

current_atom/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

predicate_property/3: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

predicate_property/2: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

time_option/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

tick_option/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

clockfreq_option/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

memory_option/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

garbage_collection_option/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

symbol_option/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

time_result/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

memory_result/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

gc_result/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

symbol_result/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

new_atom/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

tick_result/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

clockfreq_result/1: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

statistics/2: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

statistics/0: (UNDOC_REEXPORT)
Imported from `prolog_sys` (see the corresponding documentation for details).

close_file/1: (UNDOC_REEXPORT)
Imported from `dec10_io` (see the corresponding documentation for details).

told/0: (UNDOC_REEXPORT)
Imported from `dec10_io` (see the corresponding documentation for details).

telling/1: (UNDOC_REEXPORT)
Imported from `dec10_io` (see the corresponding documentation for details).

tell/1: (UNDOC_REEXPORT)
Imported from `dec10_io` (see the corresponding documentation for details).

seen/0: (UNDOC_REEXPORT)
Imported from `dec10_io` (see the corresponding documentation for details).

seeing/1: (UNDOC_REEXPORT)
Imported from `dec10_io` (see the corresponding documentation for details).

see/1: (UNDOC_REEXPORT)
Imported from `dec10_io` (see the corresponding documentation for details).

current_key/2: (UNDOC_REEXPORT)
Imported from `old_database` (see the corresponding documentation for details).

recorded/3: (UNDOC_REEXPORT)
Imported from `old_database` (see the corresponding documentation for details).

recordz/3: (UNDOC_REEXPORT)
Imported from `old_database` (see the corresponding documentation for details).

recorda/3: (UNDOC_REEXPORT)
Imported from `old_database` (see the corresponding documentation for details).

ttydisplay_string/1: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttyskipeol/0: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttydisplayq/1: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttydisplay/1: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttyflush/0: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttytab/1: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttyskip/1: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttyput/1: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttynl/0: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttyget1/1: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

ttyget/1: (UNDOC_REEXPORT)
Imported from `ttyout` (see the corresponding documentation for details).

PART III - ISO-Prolog library (iso)

Author(s): The CLIP Group.

This part documents the *iso* package which provides to Ciao programs (most of) the ISO-Prolog functionality, including the *ISO-Prolog builtins* not covered by the basic library.

32 ISO-Prolog package

Author(s): The CLIP Group.

This library package allows the use of the ISO-Prolog predicates in Ciao programs. The compatibility is not at 100% yet.

32.1 Usage and interface (`iso_doc`)

- **Library usage:**
  `:- use_package(iso).`
  or
  `:- module(...,...,[iso]).`
- **New operators defined:**
  `-->/2` [1200,xfx], `|/2` [1100,xfy].
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`.

33 All solutions predicates

Author(s): Richard A. O'Keefe (first version), David H.D. Warren (first version), Mats Carlsson (changes), Daniel Cabeza, Manuel Hermenegildo.

This module implements the standard solution aggregation predicates.

When there are many solutions to a problem, and when all those solutions are required to be collected together, this can be achieved by repeatedly backtracking and gradually building up a list of the solutions. The following built-in predicates are provided to automate this process.

33.1 Usage and interface (aggregates)

- **Library usage:**
  `:- use_module(library(aggregates)).`
- **Exports:**
  - *Predicates:*
    `setof/3`, `bagof/3`, `findall/3`, `findall/4`, `findnsols/4`, `findnsols/5`, `^/2`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`, `sort`, `lists`.

33.2 Documentation on exports (aggregates)

setof/3: PREDICATE

`setof(Template,Generator,Set)`

Finds the `Set` of instances of the `Template` satisfying `Generator`. The set is in ascending order (see Chapter 18 [Comparing terms], page 135 for a definition of this order) without duplicates, and is non-empty. If there are no solutions, `setof` fails. `setof` may succeed in more than one way, binding free variables in `Generator` to different values. This can be avoided by using existential quantifiers on the free variables in front of `Generator`, using `^/2`. For example, given the clauses:

```
father(bill, tom).
father(bill, ann).
father(bill, john).
father(harry, july).
father(harry, daniel).
```

The following query produces two alternative solutions via backtracking:

```
?- setof(X,father(F,X),Sons).

F = bill,
Sons = [ann,john,tom] ? ;

F = harry,
Sons = [daniel,july] ? ;

no
```

```
?-
```

Meta-predicate with arguments: `setof(?,goal,?)`.

General properties: `setof(X,Y,Z)`

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP as `findall(X,Y,Z)`. (basic_props:native/2)

Usage: ⟨• ISO •⟩

– *Call and exit should be compatible with:*

`Set` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Template` is any term. (basic_props:term/1)

`Generator` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Template` is any term. (basic_props:term/1)

`Set` is a list. (basic_props:list/1)

– *The following properties should hold globally:*

`Template` is not further instantiated. (basic_props:not_further_inst/2)

bagof/3: PREDICATE

`bagof(Template,Generator,Bag)`

Finds all the instances of the `Template` produced by the `Generator`, and returns them in the `Bag` in the order in which they were found. If the `Generator` contains free variables which are not bound in the `Template`, it assumes that this is like any other Prolog question and that you want bindings for those variables. This can be avoided by using existential quantifiers on the free variables in front of the `Generator`, using `^/2`.

Meta-predicate with arguments: `bagof(?,goal,?)`.

General properties: `bagof(X,Y,Z)`

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP as `findall(X,Y,Z)`. (basic_props:native/2)

Usage: ⟨• ISO •⟩

– *Call and exit should be compatible with:*

`Bag` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Template` is any term. (basic_props:term/1)

`Generator` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Template` is any term. (basic_props:term/1)

`Bag` is a list. (basic_props:list/1)

– *The following properties should hold globally:*

`Template` is not further instantiated. (basic_props:not_further_inst/2)

findall/3: PREDICATE

`findall(Template,Generator,List)`

A special case of bagof, where all free variables in the `Generator` are taken to be existentially quantified. Faster than the other aggregation predicates.

Meta-predicate with arguments: `findall(?,goal,?)`.

Usage: ⟨• ISO •⟩

- *Calls should, and exit will be compatible with:*

  `List` is a list. (basic_props:list/1)

- *The following properties should hold at call time:*

  `Template` is any term. (basic_props:term/1)

  `Generator` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

- *The following properties hold upon exit:*

  `Template` is any term. (basic_props:term/1)

  `List` is a list. (basic_props:list/1)

- *The following properties hold globally:*

  `Template` is not further instantiated. (basic_props:not_further_inst/2)

  This predicate is understood natively by CiaoPP. (basic_props:native/1)

  All the calls of the form `findall(Template,Generator,List)` do not fail. (native_props:not_fails/1)

  All calls of the form `findall(Template,Generator,List)` are deterministic. (native_props:is_det/1)

findall/4: PREDICATE

Meta-predicate with arguments: `findall(?,goal,?,?)`.

Usage:

- *Description:* As `findall/3`, but returning in `Tail` the tail of `List` (findall(`Template`, `Generator`, `List`, `Tail`)).

- *Call and exit should be compatible with:*

  `Arg3` is any term. (basic_props:term/1)

  `Arg4` is any term. (basic_props:term/1)

- *The following properties should hold at call time:*

  `Arg1` is any term. (basic_props:term/1)

  `Arg2` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

- *The following properties should hold upon exit:*

  `Arg1` is any term. (basic_props:term/1)

  `Arg3` is any term. (basic_props:term/1)

  `Arg4` is any term. (basic_props:term/1)

- *The following properties should hold globally:*

  `Arg1` is not further instantiated. (basic_props:not_further_inst/2)

findnsols/4: PREDICATE

`findnsols(N,Template,Generator,List)`

As `findall/3`, but generating at most `N` solutions of `Generator`. Thus, the length of `List` will not be greater than `N`. If `N=<0`, returns directly an empty list. This predicate is especially useful if `Generator` may have an infinite number of solutions.

Meta-predicate with arguments: `findnsols(?,?,goal,?)`.

Usage:

- *Call and exit should be compatible with:*
  `List` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*
  `N` is an integer. (basic_props:int/1)
  `Template` is any term. (basic_props:term/1)
  `Generator` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties should hold upon exit:*
  `Template` is any term. (basic_props:term/1)
  `List` is a list. (basic_props:list/1)
- *The following properties should hold globally:*
  `Template` is not further instantiated. (basic_props:not_further_inst/2)

findnsols/5: PREDICATE

`findnsols(N,Template,Generator,List,Tail)`

As `findnsols/4`, but returning in `Tail` the tail of `List`.

Meta-predicate with arguments: `findnsols(?,?,goal,?,?)`.

Usage:

- *The following properties should hold at call time:*
  `N` is an integer. (basic_props:int/1)
  `Template` is any term. (basic_props:term/1)
  `Generator` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties should hold upon exit:*
  `Template` is any term. (basic_props:term/1)
- *The following properties should hold globally:*
  `Template` is not further instantiated. (basic_props:not_further_inst/2)

^/2: PREDICATE

Meta-predicate with arguments: `(?)^goal`.

General properties: `_X^Y`

- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP as `call(Y)`. (basic_props:native/2)

Usage: `X^P`

- *Description:* Existential quantification: `X` is existentially quantified in `P`. E.g., in `A^p(A,B)`, `A` is existentially quantified. Used only within aggregation predicates. In all other contexts, simply, execute the procedure call `P`.

– *The following properties should hold at call time:*
`X` is a free variable. (term_typing:var/1)
`P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

33.3 Known bugs and planned improvements (`aggregates`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

34 Dynamic predicates

Author(s): Daniel Cabeza, The CLIP Group.

The package `dynamic_clauses` provides the assert/retract family of predicates to manipulate dynamic predicates.

The defined predicates allow modification of the program as it is actually running. Clauses can be added to the program (*asserted*) or removed from the program (*retracted*), as well as inspected. Note that in Ciao only the dynamic predicates of the current module (or accessible dynamic multifile predicates) can be accessed and modified. This limits the bad impact to global analysis of this dynamic modification of the program. Thus, if dynamic predicates are exported, to be able to inspect or modify them externally some accessing predicates need to be implemented and exported alongside.

For the inspecting/manipulating predicates, the argument which corresponds to the clause head must be instantiated to an atom or a compound term. The argument corresponding to the clause must be instantiated either to a term `Head :- Body` or, if the body part is empty, to `Head`. An empty body part is represented as `true`.

Note that using this library is very detrimental to global analysis, and that for most uses the predicates listed in Chapter 25 [Fast/concurrent update of facts], page 187 suffice.

34.1 Usage and interface (`dynamic_rt`)

- **Library usage:**

  To be able to handle dynamic predicates in a module, load the library package `dynamic_clauses`, either by putting it in the package list of the module or using the `use_package/1` directive. Do not load directly the `dynamic_rt` module.
- **Exports:**
  - *Predicates:*

    `asserta/1`, `asserta/2`, `assertz/1`, `assertz/2`, `assert/1`, `assert/2`, `retract/1`, `retractall/1`, `abolish/1`, `clause/2`, `mfclause/2`, `current_predicate/1`, `current_predicate/2`, `dynamic/1`, `data/1`, `wellformed_body/3`.
  - *Multifiles:*

    `do_on_abolish/1`.
- **Other modules used:**
  - *System library modules:*

    `prolog_sys`, `iso_misc`.

34.2 Documentation on exports (`dynamic_rt`)

asserta/1: PREDICATE

Usage: `asserta(Clause)` ⟨• ISO •⟩

- *Description:* The current instance of `Clause` is interpreted as a clause and is added to the current program. The predicate concerned must be dynamic. The new clause becomes the *first* clause for the predicate concerned. Any uninstantiated variables in `Clause` will be replaced by new private variables.

– *The following properties should hold at call time:*
`Clause` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Clause` is a well-formed clause (dynamic_rt:clause/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

asserta/2: PREDICATE

Usage: `asserta(Clause,Ref)`

– *Description:* Like `asserta/1`. `Ref` is a unique identifier of the asserted clause.
– *The following properties should hold at call time:*
`Clause` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Ref` is a free variable. (term_typing:var/1)
`Clause` is a well-formed clause (dynamic_rt:clause/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

assertz/1: PREDICATE

Usage: `assertz(Clause)` ⟨• ISO •⟩

– *Description:* Like `asserta/1`, except that the new clause becomes the *last* clause for the predicate concerned.
– *The following properties should hold at call time:*
`Clause` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Clause` is a well-formed clause (dynamic_rt:clause/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

assertz/2: PREDICATE

Usage: `assertz(Clause,Ref)`

– *Description:* Like `assertz/1`. `Ref` is a unique identifier of the asserted clause.
– *The following properties should hold at call time:*
`Clause` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Ref` is a free variable. (term_typing:var/1)
`Clause` is a well-formed clause (dynamic_rt:clause/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

assert/1: PREDICATE

Usage: `assert(Clause)`

– *Description:* Identical to `assertz/1`. Included for compatibility.

– *The following properties should hold at call time:*

`Clause` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Clause` is a well-formed clause (dynamic_rt:clause/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

assert/2: PREDICATE

Usage: `assert(Clause,Ref)`

– *Description:* Identical to `assertz/2`. Included for compatibility.

– *The following properties should hold at call time:*

`Clause` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Ref` is a free variable. (term_typing:var/1)

`Clause` is a well-formed clause (dynamic_rt:clause/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

retract/1: PREDICATE

Usage: `retract(Clause)` ⟨• ISO •⟩

– *Description:* The first clause in the program that matches `Clause` is erased. The predicate concerned must be dynamic.

The predicate `retract/1` may be used in a non-determinate fashion, i.e., it will successively retract clauses matching the argument through backtracking. If reactivated by backtracking, invocations of the predicate whose clauses are being retracted will proceed unaffected by the retracts. This is also true for invocations of `clause` for the same predicate. The space occupied by a retracted clause will be recovered when instances of the clause are no longer in use.

– *The following properties should hold at call time:*

`Clause` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Clause` is a well-formed clause (dynamic_rt:clause/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

retractall/1: PREDICATE

Usage: `retractall(Head)`

– *Description:* Erase all clauses whose head matches `Head`, where `Head` must be instantiated to an atom or a compound term. The predicate concerned must be dynamic. The predicate definition is retained.

– *The following properties should hold at call time:*

`Head` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Head` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

abolish/1: PREDICATE

Meta-predicate with arguments: `abolish(spec)`.

Usage: `abolish(Spec)` ⟨• ISO •⟩

– *Description:* Erase all clauses of the predicate specified by the predicate spec `Spec`. The predicate definition itself is also erased (the predicate is deemed undefined after execution of the abolish). The predicates concerned must all be user defined.

– *The following properties should hold at call time:*

`Spec` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Spec` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

clause/2: PREDICATE

Usage: `clause(Head,Body)` ⟨• ISO •⟩

– *Description:* The clause '`Head :- Body`' exists in the current module. The predicate concerned must be dynamic.

– *The following properties should hold at call time:*

`Head` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Head` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Body` is a clause body (dynamic_rt:body/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

mfclause/2: PREDICATE

Usage: `mfclause(Head,Body)` ⟨• ISO •⟩

– *Description:* There is a clause '`Head :- Body`' of a dynamic multifile predicate accessible from this module.

– *The following properties should hold at call time:*

`Head` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Head` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`Body` is a clause body (dynamic_rt:body/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_predicate/1: PREDICATE

Usage: `current_predicate(Spec)` ⟨• ISO •⟩

– *Description:* A predicate in the current module is named `Spec`.

– *The following properties should hold upon exit:*

`Spec` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_predicate/2: PREDICATE

Usage: `current_predicate(Spec,Module)`

– *Description:* A predicate in `Module` is named `Spec`. `Module` never is an engine module.

– *The following properties should hold upon exit:*

`Spec` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

`Module` is an internal module identifier (system_info:internal_module_id/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

dynamic/1: PREDICATE

`dynamic Spec`

`Spec` is of the form `F/A`. The predicate named `F` with arity `A` is made dynamic in the current module at runtime (useful for predicate names generated on-the-fly). If the predicate functor name `F` is uninstatiated, a new, unique, predicate name is generated at runtime.

Usage: `dynamic Spec`

– *The following properties should hold at call time:*

`Spec` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Spec` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

data/1: PREDICATE

`data Spec`

`Spec` is of the form `F/A`. The predicate named `F` with arity `A` is made data in the current module at runtime (useful for predicate names generated on-the-fly). If the predicate functor name `F` is uninstatiated, a new, unique, predicate name is generated at runtime.

Usage: `data Spec`

– *The following properties should hold at call time:*
`Spec` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Spec` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

erase/1: PREDICATE

`erase(Ref)`

Deletes the clause referenced by `Ref`, the identifier obtained by using `asserta/2` or `assertz/2`.

wellformed_body/3: PREDICATE

`wellformed_body(BodyIn,Env,BodyOut)`

`BodyIn` is a well-formed clause body. `BodyOut` is its counterpart with no single-variable meta-goals (i.e., with `call(X)` for `X`). `Env` denotes if global cuts are admissible in `BodyIn` (`+` if they are, `-` if they are not).

34.3 Documentation on multifiles (dynamic_rt)

do_on_abolish/1: PREDICATE

`do_on_abolish(Head)`

A hook predicate which will be called when the definition of the predicate of `Head` is abolished.

The predicate is *multifile*.

Usage: `do_on_abolish(G)`

– *The following properties should hold at call time:*
`G` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

34.4 Known bugs and planned improvements (dynamic_rt)

- Package dynamic_clauses cannot be loaded into the shell.

35 Term input

Author(s): Daniel Cabeza (modifications and documentation, adapted from SICStus 0.6 code), Manuel Carro (modifications and documentation).

This module provides falicities to read terms in Prolog syntax. This is very convenient in many cases (and not only if you are writing a Prolog compiler), because Prolog terms are easy to write and can convey a lot of information in a human-readable fashion.

35.1 Usage and interface (`read`)

- **Library usage:**
  `:- use_module(library(read)).`
- **Exports:**
  - *Predicates:*
    `read/1`, `read/2`, `read_term/2`, `read_term/3`, `read_top_level/3`, `second_prompt/2`.
  - *Regular Types:*
    `read_option/1`.
  - *Multifiles:*
    `define_flag/3`.
- **Other modules used:**
  - *System library modules:*
    `tokenize`, `operators`, `lists`.

35.2 Documentation on exports (`read`)

read/1: PREDICATE

`read(Term)`

Like `read(Stream,Term)` with `Stream` associated to the current input stream.

Usage: ⟨• ISO •⟩

- *Call and exit should be compatible with:*
  `Term` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*
  `Term` is any term. (basic_props:term/1)

read/2: PREDICATE

Usage: `read(Stream,Term)` ⟨• ISO •⟩

- *Description:* The next term, delimited by a full-stop (i.e., a `.` followed by either a space or a control character), is read from `Stream` and is unified with `Term`. The syntax of the term must agree with current operator declarations. If the end of `Stream` has been reached, `Term` is unified with the term `end_of_file`. Further calls to `read/2` for the same stream will then cause an error, unless the stream is connected to the terminal (in which case a prompt is opened on the terminal).

– *The following properties should hold at call time:*

`Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Stream` is an open stream. (streams_basic:stream/1)

`Term` is any term. (basic_props:term/1)

read_term/2: PREDICATE

Usage: `read_term(Term,Options)` ⟨• ISO •⟩

– *Description:* Like `read_term/3`, but reading from the current input

– *The following properties should hold at call time:*

`Options` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Term` is any term. (basic_props:term/1)

`Options` is a list of `read_option`s. (basic_props:list/2)

read_term/3: PREDICATE

Usage: `read_term(Stream,Term,Options)` ⟨• ISO •⟩

– *Description:* Reads a `Term` from `Stream` with the ISO-Prolog `Options`. These options can control the behavior of read term (see `read_option/1`).

– *The following properties should hold at call time:*

`Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Options` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Stream` is an open stream. (streams_basic:stream/1)

`Term` is any term. (basic_props:term/1)

`Options` is a list of `read_option`s. (basic_props:list/2)

read_top_level/3: PREDICATE

`read_top_level(Stream,Data,Variables)`

Predicate used to read in the Top Level.

second_prompt/2: PREDICATE

Usage: `second_prompt(Old,New)`

– *Description:* Changes the prompt (the *second prompt*, as opposed to the first one, used by the toplevel) used by `read/2` and friends to `New`, and returns the current one in `Old`.

– *The following properties should hold upon exit:*

`Old` is currently instantiated to an atom. (term_typing:atom/1)

`New` is currently instantiated to an atom. (term_typing:atom/1)

read_option/1: REGTYPE

Usage: `read_option(Option)`

– *Description:* `Option` is an allowed `read_term/[2,3]` option. These options are:

```
read_option(variables(_V)).
read_option(variable_names(_N)).
read_option(singletons(_S)).
read_option(lines(_StartLine,_EndLine)).
read_option(dictionary(_Dict)).
```

They can be used to return the singleton variables in the term, a list of variables, etc.

– *The following properties should hold upon exit:*

`Option` is currently instantiated to an atom. (term_typing:atom/1)

35.3 Documentation on multifiles (`read`)

define_flag/3: PREDICATE

Defines flags as follows:

```
define_flag(read_hiord,[on,off],off).
define_flag(read_curly_blocks,[on,off],off).
define_flag(read_postfix_blocks,[on,off],off).
```

(See Chapter 24 [Changing system behaviour and various flags], page 181).

If flag is `on` (it is `off` by default), a variable followed by a parenthesized lists of arguments is read as a `call/N` term, except if the variable is anonymous, in which case it is read as an anonymous predicate abstraction head. For example, `P(X)` is read as `call(P,X)` and `_(X,Y)` as `''(X,Y)`.

The predicate is *multifile*.

Usage: `define_flag(Flag,FlagValues,Default)`

– *The following properties hold upon exit:*

`Flag` is an atom. (basic_props:atm/1)

Define the valid flag values (basic_props:flag_values/1)

35.4 Known bugs and planned improvements (`read`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.
- The comma cannot be redefined as an operator, it is defined in any case as op(1000, xfy,[',']).

36 Term output

Author(s): Richard A. O'Keefe (original version), Mats Carlsson (changes), Daniel Cabeza (changes), Manuel Hermenegildo (changes), Manuel Carro (changes).

This library provides different predicates for term output, additional to the kernel predicates `display/1`- `display/2` and `displayq/1`-`displayq/2`. All the predicates defined in ISO-Prolog are included, plus other traditionally provided by Prolog Implementations. Output predicates are provided in two versions: one that uses the current output stream and other in which the stream is specified explicitly, as an additional first argument.

36.1 Usage and interface (`write`)

- **Library usage:**
  `:- use_module(library(write)).`
- **Exports:**
  - *Predicates:*
    `write_term/3`, `write_term/2`, `write/2`, `write/1`, `writeq/2`, `writeq/1`, `write_canonical/2`, `write_canonical/1`, `print/2`, `print/1`, `write_list1/1`, `portray_clause/2`, `portray_clause/1`, `numbervars/3`, `prettyvars/1`, `printable_char/1`, `write_attribute/1`.
  - *Properties:*
    `write_option/1`.
  - *Multifiles:*
    `define_flag/3`, `portray_attribute/2`, `portray/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`, `operators`, `sort`.

36.2 Documentation on exports (`write`)

write_term/3: PREDICATE

Usage: `write_term(Stream,Term,OptList)` ⟨• ISO •⟩

- *Description:* Outputs the term `Term` to the stream `Stream`, with the list of write-options `OptList`. See `write_option/1` type for default options.
- *The following properties should hold at call time:*
  `OptList` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Stream` is an open stream. (streams_basic:stream/1)
  `Term` is any term. (basic_props:term/1)
  `OptList` is a list of `write_option`s. (basic_props:list/2)
- *The following properties should hold globally:*
  `Stream` is not further instantiated. (basic_props:not_further_inst/2)

write_term/2: PREDICATE

Usage: `write_term(Term,OptList)` ⟨• ISO •⟩

– *Description:* Behaves like `current_output(S), write_term(S,Term,OptList)`.

– *The following properties should hold at call time:*

`OptList` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Term` is any term. (basic_props:term/1)

`OptList` is a list of `write_option`s. (basic_props:list/2)

write_option/1: PROPERTY

`Opt` is a valid write option which affects the predicate `write_term/3` and similar ones. Possible write_options are:

- **quoted(*bool*):** If *bool* is `true`, atoms and functors that can't be read back by `read_term/3` are quoted, if it is `false`, each atom and functor is written as its name. Default value is `false`.
- **ignore_ops(*flag*):** If *flag* is `true`, each compound term is output in functional notation, if it is `ops`, curly bracketed notation and list notation is enabled when outputing compound terms, if it is `false`, also operator notation is enabled when outputing compound terms. Default value is `false`.
- **numbervars(*bool*):** If *bool* is `true`, a term of the form `'$VAR'(N)` where `N` is an integer, is output as a variable name consisting of a capital letter possibly followed by an integer, a term of the form `'$VAR'(Atom)` where `Atom` is an atom, as this atom (without quotes), and a term of the form `'$VAR'(String)` where `String` is a character string, as the atom corresponding to this character string. See predicates `numbervars/3` and `prettyvars/1`. If *bool* is `false` this cases are not treated in any special way. Default value is `false`.
- **portrayed(*bool*):** If *bool* is `true`, then call multifile predicates `portray/1` and `portray_attribute/2`, to provide the user handlers for pretty printing some terms. `portray_attribute/2` is called whenever an attributed variable is to be printed, `portray/1` is called whenever a non-variable term is to be printed. If either call succeeds, then it is assumed that the term has been output, else it is printed as usual. If *bool* is `false`, these predicates are not called. Default value is `false`. This option is set by the toplevel when writting the final values of variables, and by the debugging package when writting the goals in the tracing messages. Thus you can vary the forms of these messages if you wish.
- **max_depth(*depth*):** *depth* is a positive integer or cero. If it is positive, it denotes the depth limit on printing compound terms. If it is cero, there is no limit. Default value is `0` (no limit).
- **priority(*prio*):** *prio* is an integer between 1 and 1200. If the term to be printed has higher priority than *prio*, it will be printed parenthesized. Default value is 1200 (no term parenthesized).

.

Usage: `write_option(Opt)`

– *Description:* `Opt` is a valid write option.

write/2: PREDICATE

Usage: `write(Stream,Term)` ⟨• ISO •⟩

– *Description:* Behaves like `write_term(Stream, Term, [numbervars(true)])`.

– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
`Term` is any term. (basic_props:term/1)
– *The following properties hold globally:*
`Stream` is not further instantiated. (basic_props:not_further_inst/2)
All calls of the form `write(Stream,Term)` are deterministic. (native_props:is_det/1)

write/1: PREDICATE
Usage: `write(Term)` ⟨• ISO •⟩
– *Description:* Behaves like `current_output(S), write(S,Term)`.
– *The following properties should hold at call time:*
`Term` is any term. (basic_props:term/1)
– *The following properties hold globally:*
All calls of the form `write(Term)` are deterministic. (native_props:is_det/1)

writeq/2: PREDICATE
Usage: `writeq(Stream,Term)` ⟨• ISO •⟩
– *Description:* Behaves like `write_term(Stream, Term, [quoted(true), numbervars(true)])`.
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
`Term` is any term. (basic_props:term/1)
– *The following properties hold globally:*
`Stream` is not further instantiated. (basic_props:not_further_inst/2)
All calls of the form `writeq(Stream,Term)` are deterministic. (native_props:is_det/1)

writeq/1: PREDICATE
Usage: `writeq(Term)` ⟨• ISO •⟩
– *Description:* Behaves like `current_output(S), writeq(S,Term)`.
– *The following properties should hold at call time:*
`Term` is any term. (basic_props:term/1)
– *The following properties hold globally:*
All calls of the form `writeq(Term)` are deterministic. (native_props:is_det/1)

write_canonical/2: PREDICATE
Usage: `write_canonical(Stream,Term)` ⟨• ISO •⟩
– *Description:* Behaves like `write_term(Stream, Term, [quoted(true), ignore_ops(true)])`. The output of this predicate can always be parsed by `read_term/2` even if the term contains special characters or if operator declarations have changed.
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
`Term` is any term. (basic_props:term/1)

– *The following properties should hold globally:*
`Stream` is not further instantiated. (basic_props:not_further_inst/2)

write_canonical/1: PREDICATE
Usage: `write_canonical(Term)` ⟨• ISO •⟩
– *Description:* Behaves like `current_output(S), write_canonical(S,Term)`.
– *The following properties should hold at call time:*
`Term` is any term. (basic_props:term/1)

print/2: PREDICATE
Usage: `print(Stream,Term)`
– *Description:* Behaves like `write_term(Stream, Term, [numbervars(true), portrayed(true)])`.
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
`Term` is any term. (basic_props:term/1)
– *The following properties should hold globally:*
`Stream` is not further instantiated. (basic_props:not_further_inst/2)

print/1: PREDICATE
Usage: `print(Term)`
– *Description:* Behaves like `current_output(S), print(S,Term)`.
– *The following properties should hold at call time:*
`Term` is any term. (basic_props:term/1)

write_list1/1: PREDICATE
Usage:
– *Description:* Writes a list to current output one element in each line.
– *The following properties should hold at call time:*
`Arg1` is a list. (basic_props:list/1)

portray_clause/2: PREDICATE
Usage: `portray_clause(Stream,Clause)`
– *Description:* Outputs the clause `Clause` onto `Stream`, pretty printing its variables and using indentation, including a period at the end. This predicate is used by `listing/0`.
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
`Clause` is any term. (basic_props:term/1)
– *The following properties should hold globally:*
`Stream` is not further instantiated. (basic_props:not_further_inst/2)

portray_clause/1: PREDICATE

Usage: `portray_clause(Clause)`

- *Description:* Behaves like `current_output(S), portray_clause(S,Term).`
- *The following properties should hold at call time:*

  `Clause` is any term. (basic_props:term/1)

numbervars/3: PREDICATE

Usage: `numbervars(Term,N,M)`

- *Description:* Unifies each of the variables in term `Term` with a term of the form `'$VAR'(I)` where `I` is an integer from `N` onwards. `M` is unified with the last integer used plus 1. If the resulting term is output with a write option `numbervars(true)`, in the place of the variables in the original term will be printed a variable name consisting of a capital letter possibly followed by an integer. When `N` is 0 you will get the variable names A, B, ..., Z, A1, B1, etc.
- *The following properties should hold at call time:*

  `N` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Term` is any term. (basic_props:term/1)

  `N` is an integer. (basic_props:int/1)

  `M` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*

  `Term` is any term. (basic_props:term/1)

  `N` is an integer. (basic_props:int/1)

  `M` is an integer. (basic_props:int/1)

prettyvars/1: PREDICATE

Usage: `prettyvars(Term)`

- *Description:* Similar to `numbervars(Term,0,_)`, except that singleton variables in `Term` are unified with `'$VAR'('_')`, so that when the resulting term is output with a write option `numbervars(true)`, in the place of singleton variables `_` is written. This predicate is used by `portray_clause/2`.
- *The following properties should hold at call time:*

  `Term` is any term. (basic_props:term/1)

printable_char/1: PREDICATE

Usage: `printable_char(Char)`

- *Description:* `Char` is the code of a character which can be printed.
- *The following properties should hold at call time:*

  `Char` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Char` is an integer which is a character code. (basic_props:character_code/1)

write_attribute/1: PREDICATE

No further documentation available for this predicate.

36.3 Documentation on multifiles (`write`)

define_flag/3: PREDICATE

Defines flags as follows:

```
define_flag(write_strings,[on,off],off).
```

(See Chapter 24 [Changing system behaviour and various flags], page 181).

If flag is `on`, lists which may be written as strings are.

The predicate is *multifile*.

Usage: `define_flag(Flag,FlagValues,Default)`

– *The following properties hold upon exit:*

`Flag` is an atom. (basic_props:atm/1)

Define the valid flag values (basic_props:flag_values/1)

portray_attribute/2: PREDICATE

The predicate is *multifile*.

Usage: `portray_attribute(Attr,Var)`

– *Description: A user defined predicate.* When an attributed variable `Var` is about to be printed, this predicate receives the variable and its attribute `Attr`. The predicate should either print something based on `Attr` or `Var`, or do nothing and fail. In the latter case, the default printer (`write/1`) will print the attributed variable like an unbound variable, e.g. `_673`.

– *The following properties should hold at call time:*

`Attr` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Var` is a free variable. (term_typing:var/1)

portray/1: PREDICATE

The predicate is *multifile*.

Usage: `portray(T)`

– *Description: A user defined predicate.* This should either print the `Term` and succeed, or do nothing and fail. In the latter case, the default printer (`write/1`) will print the `Term`.

– *The following properties should hold at call time:*

`T` is any term. (basic_props:term/1)

– *The following properties hold upon exit:*

`T` is any term. (basic_props:term/1)

36.4 Known bugs and planned improvements (`write`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

37 Defining operators

Author(s): Daniel Cabeza (modifications and documentation, adapted from SICStus 0.6 code), Manuel Carro (modifications and documentation).

Operators allow writting terms in a more clear way than the standard functional notation. Standard operators in Ciao are defined by this predicate (but note that the compiler itself defines more operators at compile time):

```
standard_ops :-
        op(1200,xfx,[:-]),
        op(1200,fx,[:-,?-]),
        op(1100,xfy,[;]),
        op(1050,xfy,[->]),
        op(1000,xfy,[',']),
        op(900,fy,[\+]),
        op(700,xfx,[=,\=,==,\==,@<,@>,@=<,@>=,=..,is,=:=,=\=,<,=<,>,>=]),
        op(550,xfx,[:]),
        op(500,yfx,[+,-,/\,\/,#]),
        op(500,fy,[++,--]),
        op(400,yfx,[*,/,//,rem,mod,<<,>>]),
        op(200,fy,[+,-,\]),
        op(200,xfx,[**]),
        op(200,xfy,[^]).
```

37.1 Usage and interface (operators)

- **Library usage:**
  `:- use_module(library(operators)).`
- **Exports:**
  - *Predicates:*
    `op/3`,
    `current_op/3`, `current_prefixop/3`, `current_infixop/4`, `current_postfixop/3`, `standard_ops/0`.

37.2 Documentation on exports (operators)

op/3: PREDICATE

`op(Precedence,Type,Name)`

Declares the atom `Name` to be an operator of the stated `Type` and `Precedence` (0 =< `Precedence` =< 1200). `Name` may also be a list of atoms in which case all of them are declared to be operators. If `Precedence` is 0 then the operator properties of `Name` (if any) are cancelled. Note that, unlike in ISO-Prolog, it is allowed to define two operators with the same name, one infix and the other postfix.

Usage: ⟨• ISO •⟩

- *The following properties should hold at call time:*
  `Precedence` is an integer. (basic_props:int/1)

`Type` specifies the type and associativity of an operator. (basic_props:operator_specifier/1)

`Name` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_op/3: PREDICATE

`current_op(Precedence,Type,Op)`

The atom `Op` is currently an operator of type `Type` and precedence `Precedence`. Neither `Op` nor the other arguments need be instantiated at the time of the call; i.e., this predicate can be used to generate as well as to test.

Usage: ⟨• ISO •⟩

– *Call and exit should be compatible with:*

`Precedence` is an integer. (basic_props:int/1)

`Type` specifies the type and associativity of an operator. (basic_props:operator_specifier/1)

`Op` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*

`Precedence` is an integer. (basic_props:int/1)

`Type` specifies the type and associativity of an operator. (basic_props:operator_specifier/1)

`Op` is an atom. (basic_props:atm/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_prefixop/3: PREDICATE

`current_prefixop(Op,Less,Precedence)`

Similar to `current_op/3`, but it concerns only the prefix operators. It returns **only one solution**. Not a predicate for general use.

Usage:

– *Call and exit should be compatible with:*

`Op` is an atom. (basic_props:atm/1)

`Less` is an integer. (basic_props:int/1)

`Precedence` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`Op` is an atom. (basic_props:atm/1)

`Less` is an integer. (basic_props:int/1)

`Precedence` is an integer. (basic_props:int/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_infixop/4: PREDICATE

`current_infixop(Op,LeftLess,Prec,RightLess)`

Similar to `current_op/3`, but it concerns only infix operators. It returns **only one solution**. Not a predicate for general use.

Usage:

– *Call and exit should be compatible with:*

`Op` is an atom. (basic_props:atm/1)

`LeftLess` is an integer. (basic_props:int/1)

`Prec` is an integer. (basic_props:int/1)

`RightLess` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`Op` is an atom. (basic_props:atm/1)

`LeftLess` is an integer. (basic_props:int/1)

`Prec` is an integer. (basic_props:int/1)

`RightLess` is an integer. (basic_props:int/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_postfixop/3: PREDICATE

`current_postfixop(Op,Less,Precedence)`

Similar to `current_op/3`, but it concerns only the postfix operators. It returns **only one solution**. Not a predicate for general use.

Usage:

– *Call and exit should be compatible with:*

`Op` is an atom. (basic_props:atm/1)

`Less` is an integer. (basic_props:int/1)

`Precedence` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`Op` is an atom. (basic_props:atm/1)

`Less` is an integer. (basic_props:int/1)

`Precedence` is an integer. (basic_props:int/1)

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

standard_ops/0: PREDICATE

No further documentation available for this predicate.

38 The Iso Byte Char module

Author(s): The CLIP Group, Daniel Cabeza, Edison Mera (documentation), Manuel Hermenegildo (minor mods).

This module provides some basic predicates according to the ISO specification of byte and char manipulation.

38.1 Usage and interface (iso_byte_char)

- **Library usage:**
  `:- use_module(library(iso_byte_char)).`
- **Exports:**
  - *Predicates:*
    `char_code/2`, `atom_chars/2`, `number_chars/2`, `char_codes/2`, `get_byte/1`, `get_byte/2`, `peek_byte/1`, `peek_byte/2`, `put_byte/1`, `put_byte/2`, `get_char/1`, `get_char/2`, `peek_char/1`, `peek_char/2`, `put_char/1`, `put_char/2`.

38.2 Documentation on exports (iso_byte_char)

char_code/2: PREDICATE

`char_code(Char,Code)`

Succeeds iff the character code of the one char atom `Char` is `Code`.

Usage 1: ⟨• ISO •⟩

- *Call and exit should be compatible with:*
  `Code` is an integer which is a character code. (basic_props:character_code/1)
- *The following properties should hold at call time:*
  `Char` is an atom. (basic_props:atm/1)
- *The following properties should hold upon exit:*
  `Code` is an integer which is a character code. (basic_props:character_code/1)

Usage 2: ⟨• ISO •⟩

- *The following properties should hold at call time:*
  `Char` is a free variable. (term_typing:var/1)
  `Code` is an integer which is a character code. (basic_props:character_code/1)
- *The following properties should hold upon exit:*
  `Char` is an atom. (basic_props:atm/1)

atom_chars/2: PREDICATE

`atom_chars(Atom,Chars)`

Succeeds iff `Chars` is a list whose elements are the one-char atoms whose names are the successive characters of the name of atom `Atom`

Usage 1: ⟨• ISO •⟩

– *Call and exit should be compatible with:*
`Chars` is a list of `atm`s. (basic_props:list/2)
– *The following properties should hold at call time:*
`Atom` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
`Chars` is a list of `atm`s. (basic_props:list/2)

Usage 2: ⟨• ISO •⟩
– *The following properties should hold at call time:*
`Atom` is a free variable. (term_typing:var/1)
`Chars` is a list of `atm`s. (basic_props:list/2)
– *The following properties should hold upon exit:*
`Atom` is an atom. (basic_props:atm/1)

number_chars/2: PREDICATE

`number_chars(Number,Chars)`

Success iff `Chars` is a list whose elements are the one-char atoms corresponding to a character sequence of `Number` which could be output

Usage 1: ⟨• ISO •⟩
– *Call and exit should be compatible with:*
`Chars` is a list of `atm`s. (basic_props:list/2)
– *The following properties should hold at call time:*
`Number` is a number. (basic_props:num/1)
– *The following properties should hold upon exit:*
`Chars` is a list of `atm`s. (basic_props:list/2)

Usage 2: ⟨• ISO •⟩
– *The following properties should hold at call time:*
`Number` is a free variable. (term_typing:var/1)
`Chars` is a list of `atm`s. (basic_props:list/2)
– *The following properties should hold upon exit:*
`Number` is a number. (basic_props:num/1)

char_codes/2: PREDICATE

Usage 1:
– *Call and exit should be compatible with:*
`Arg2` is a list of `character_code`s. (basic_props:list/2)
– *The following properties should hold at call time:*
`Arg1` is a list of `atm`s. (basic_props:list/2)
– *The following properties should hold upon exit:*
`Arg2` is a list of `character_code`s. (basic_props:list/2)

Usage 2:

– *The following properties should hold at call time:*
`Arg1` is a free variable. (term_typing:var/1)
`Arg2` is a list of `character_code`s. (basic_props:list/2)
– *The following properties should hold upon exit:*
`Arg1` is a list of `atm`s. (basic_props:list/2)

get_byte/1: PREDICATE
Usage: ⟨• ISO •⟩
– *Description:* Same as `get_byte/2`, but use the current input.
– *Call and exit should be compatible with:*
`Arg1` is an integer. (basic_props:int/1)
– *The following properties should hold upon exit:*
`Arg1` is an integer. (basic_props:int/1)

get_byte/2: PREDICATE
`get_byte(Stream,Byte)`
Is true iff `Byte` unifies with the next byte to be input from the target `Stream`.
Usage: ⟨• ISO •⟩
– *Call and exit should be compatible with:*
`Byte` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties should hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)
`Byte` is an integer. (basic_props:int/1)
– *The following properties should hold globally:*
`Stream` is not further instantiated. (basic_props:not_further_inst/2)

peek_byte/1: PREDICATE
Usage: ⟨• ISO •⟩
– *Description:* Same as `peek_byte/2`, but use the current input.
– *Call and exit should be compatible with:*
`Arg1` is an integer. (basic_props:int/1)
– *The following properties should hold upon exit:*
`Arg1` is an integer. (basic_props:int/1)

peek_byte/2: PREDICATE
Usage: ⟨• ISO •⟩
– *Description:* Is true iff `Byte` unifies with the next byte to be input from the target `Stream`.

– *Call and exit should be compatible with:*
`Arg2` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Arg1` is an open stream. (streams_basic:stream/1)
– *The following properties should hold upon exit:*
`Arg1` is an open stream. (streams_basic:stream/1)
`Arg2` is an integer. (basic_props:int/1)
– *The following properties should hold globally:*
`Arg1` is not further instantiated. (basic_props:not_further_inst/2)

put_byte/1: PREDICATE

Usage: ⟨• ISO •⟩

– *Description:* Same as `put_byte/2`, but use the current input.
– *The following properties should hold at call time:*
`Arg1` is an integer. (basic_props:int/1)

put_byte/2: PREDICATE

`put_byte(Stream,Byte)`

Is true. Procedurally, `putbyte/2` is executed as follows:

a) Outputs the byte `Byte` to the target stream.

b) Changes the stream position of the target stream to take account of the byte which has been output.

c) The goal succeeds.

Usage: ⟨• ISO •⟩

– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
`Byte` is an integer. (basic_props:int/1)
– *The following properties should hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties should hold globally:*
`Stream` is not further instantiated. (basic_props:not_further_inst/2)

get_char/1: PREDICATE

Usage: ⟨• ISO •⟩

– *Description:* Same as `get_char/2`, but use the current input.
– *Call and exit should be compatible with:*
`Arg1` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
`Arg1` is an atom. (basic_props:atm/1)

get_char/2: PREDICATE

`get_char(Stream,Char)`

Is true iff `Char` unifies with the next character to be input from the target `Stream`.

Usage: ⟨• ISO •⟩

– *Call and exit should be compatible with:*
`Char` is an atom. (basic_props:atm/1)
– *The following properties should hold at call time:*
`Stream` is an open stream. (streams_basic:stream/1)
– *The following properties should hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)
`Char` is an atom. (basic_props:atm/1)
– *The following properties should hold globally:*
`Stream` is not further instantiated. (basic_props:not_further_inst/2)

peek_char/1: PREDICATE

Usage: ⟨• ISO •⟩

– *Description:* Similar to `peek_code/1`, but using char instead of code.
– *Call and exit should be compatible with:*
`Arg1` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
`Arg1` is an atom. (basic_props:atm/1)

peek_char/2: PREDICATE

Usage: ⟨• ISO •⟩

– *Description:* Similar to `peek_code/2`, but using char instead of code.
– *Call and exit should be compatible with:*
`Arg2` is an atom. (basic_props:atm/1)
– *The following properties should hold at call time:*
`Arg1` is an open stream. (streams_basic:stream/1)
– *The following properties should hold upon exit:*
`Arg1` is an open stream. (streams_basic:stream/1)
`Arg2` is an atom. (basic_props:atm/1)
– *The following properties should hold globally:*
`Arg1` is not further instantiated. (basic_props:not_further_inst/2)

put_char/1: PREDICATE

Usage: ⟨• ISO •⟩

– *Description:* Similar to `put_code/1`, but using char instead of code.
– *The following properties should hold at call time:*
`Arg1` is an atom. (basic_props:atm/1)

put_char/2: PREDICATE

Usage: ⟨• ISO •⟩

- *Description:* Similar to `put_code/2`, but using char instead of code.
- *The following properties should hold at call time:*

  `Arg1` is an open stream. (streams_basic:stream/1)

  `Arg2` is an atom. (basic_props:atm/1)
- *The following properties should hold upon exit:*

  `Arg1` is an open stream. (streams_basic:stream/1)
- *The following properties should hold globally:*

  `Arg1` is not further instantiated. (basic_props:not_further_inst/2)

39 Miscellaneous ISO Prolog predicates

Author(s): Daniel Cabeza.

This module implements some miscellaneous ISO Prolog predicates.

39.1 Usage and interface (`iso_misc`)

- **Library usage:**
  `:- use_module(library(iso_misc)).`
- **Exports:**
  - *Predicates:*
    `once/1`, `compound/1`, `sub_atom/5`, `unify_with_occurs_check/2`.
- **Other modules used:**
  - *System library modules:*
    `between`.

39.2 Documentation on exports (`iso_misc`)

once/1: PREDICATE

`once(G)`

Finds the first solution of goal `G` (if any). `once/1` behaves as `call/1`, except that no further solutions are explored on backtracking.

Meta-predicate with arguments: `once(goal)`.

Usage: ⟨• ISO •⟩

- *The following properties should hold at call time:*
  `G` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

compound/1: PREDICATE

`compound(T)`

`T` is currently instantiated to a compound term.

Usage: ⟨• ISO •⟩

- *Call and exit should be compatible with:*
  `T` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*
  `T` is any term. (basic_props:term/1)
  `T` is a compound term. (basic_props:struct/1)

sub_atom/5: PREDICATE

`sub_atom(Atom,Before,Length,After,Sub_atom)`

Is true iff atom `Atom` can be broken into three pieces, `AtomL`, `Sub_atom` and `AtomR` such that `Before` is the number of characters of the name of `AtomL`, `Length` is the number of characters of the name of `Sub_atom` and `After` is the number of characters of the name of `AtomR`

Usage: ⟨• ISO •⟩

– *Call and exit should be compatible with:*

`Before` is an integer. (basic_props:int/1)

`Length` is an integer. (basic_props:int/1)

`After` is an integer. (basic_props:int/1)

`Sub_atom` is an atom. (basic_props:atm/1)

– *The following properties should hold at call time:*

`Atom` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*

`Before` is an integer. (basic_props:int/1)

`Length` is an integer. (basic_props:int/1)

`After` is an integer. (basic_props:int/1)

`Sub_atom` is an atom. (basic_props:atm/1)

unify_with_occurs_check/2: PREDICATE

`unify_with_occurs_check(X,Y)`

Attempts to compute and apply a most general unifier of the two terms `X` and `Y`. Is true iff `X` and `Y` are unifiable.

Usage: ⟨• ISO •⟩

– *Call and exit should be compatible with:*

`X` is any term. (basic_props:term/1)

`Y` is any term. (basic_props:term/1)

– *The following properties should hold upon exit:*

`X` is any term. (basic_props:term/1)

`Y` is any term. (basic_props:term/1)

39.3 Known bugs and planned improvements (`iso_misc`)

- There is a naive implementation of compound/1, perhaps is better to implement it as a builtin – EMM.

40 Incomplete ISO Prolog predicates

Author(s): The CLIP Group.
This module implements some ISO Prolog predicates, but that are not complete yet.

40.1 Usage and interface (`iso_incomplete`)

- **Library usage:**
  `:- use_module(library(iso_incomplete)).`
- **Exports:**
  - *Predicates:*
    `close/2`, `stream_property/2`.

40.2 Documentation on exports (`iso_incomplete`)

close/2: PREDICATE

Usage:

- *The following properties should hold at call time:*
  `Arg1` is an open stream. (streams_basic:stream/1)
  iso_incomplete:close_options(Arg2) (iso_incomplete:close_options/1)
- *The following properties should hold upon exit:*
  `Arg1` is an open stream. (streams_basic:stream/1)
  iso_incomplete:close_options(Arg2) (iso_incomplete:close_options/1)
- *The following properties should hold globally:*
  `Arg1` is not further instantiated. (basic_props:not_further_inst/2)
  `Arg2` is not further instantiated. (basic_props:not_further_inst/2)

stream_property/2: PREDICATE

Usage:

- *Call and exit should be compatible with:*
  `Arg1` is an open stream. (streams_basic:stream/1)
  `Arg2` is a valid stream property. (iso_incomplete:stream_prop/1)
- *The following properties should hold upon exit:*
  `Arg1` is an open stream. (streams_basic:stream/1)
  `Arg2` is a valid stream property. (iso_incomplete:stream_prop/1)

PART IV - Classic Prolog library (classic)

Author(s): The CLIP Group.

This part documents some Ciao libraries which provide additional predicates and functionalities that, despite not being in the ISO standard, are present in many popular Prolog systems. This includes definite clause grammars (DCGs), "Quintus-style" internal database, list processing predicates, DEC-10 Prolog-style input/output, formatted output, dynamic loading of modules, activation of operators at run-time, etc.

41 Definite clause grammars

Author(s): The CLIP Group.

This library package allows the use of DCGs (Definite Clause Grammars) [Col78,PW80] in a Ciao module/program.

Definite clause grammars are an extension of the well-known context-free grammars. Prolog's grammar rules provide a convenient notation for expressing definite clause grammars. A DCG rule in Prolog takes the general form

```
head --> body.
```

meaning "a possible form for `head` is `body`". Both `body` and `head` are sequences of one or more items linked by the standard Prolog conjunction operator `","`.

Definite clause grammars extend context-free grammars in the following ways:

1. A non-terminal symbol may be any Prolog term (other than a variable or number).
2. A terminal symbol may be any Prolog term. To distinguish terminals from non-terminals, a sequence of one or more terminal symbols is written within a grammar rule as a Prolog list. An empty sequence is written as the empty list `[]`. If the terminal symbols are ASCII character codes, such lists can be written (as elsewhere) as strings. An empty sequence is written as the empty list, `[]` or `""`.
3. Extra conditions, in the form of Prolog procedure calls, may be included in the right-hand side of a grammar rule. Such procedure calls are written enclosed in `{}` brackets.
4. The left-hand side of a grammar rule consists of a non-terminal, optionally followed by a sequence of terminals (again written as a Prolog list).
5. Alternatives may be stated explicitly in the right-hand side of a grammar rule, using the disjunction operator `;`, or, also, as traditionally in Prolog, using `|` (which is treated specially when this package is loaded).
6. The cut symbol may be included in the right-hand side of a grammar rule, as in a Prolog clause. The cut symbol does not need to be enclosed in `{}` brackets.

As an example, here is a simple grammar which parses an arithmetic expression (made up of digits and operators) and computes its value.

```
expr(Z) --> term(X), "+", expr(Y), {Z is X + Y}.
expr(Z) --> term(X), "-", expr(Y), {Z is X - Y}.
expr(X) --> term(X).

term(Z) --> number(X), "*", term(Y), {Z is X * Y}.
term(Z) --> number(X), "/", term(Y), {Z is X / Y}.
term(Z) --> number(Z).

number(C) --> "+", number(C).
number(C) --> "-", number(X), {C is -X}.
number(X) --> [C], {0'0=<C, C=<0'9, X is C - 0'0}.
```

In the last rule, `C` is the ASCII code of some digit.

The query

```
?- expr(Z, "-2+3*5+1", []).
```

will compute `Z`=14. The two extra arguments are explained below.

Now, in fact, grammar rules are merely a convenient "syntactic sugar" for ordinary Prolog clauses. Each grammar rule takes an input string, analyses some initial portion, and produces the remaining portion (possibly enlarged) as output for further analysis. The arguments required for the input and output strings are not written explicitly in a grammar rule, but the syntax

implicitly defines them. We now show how to translate grammar rules into ordinary clauses by making explicit the extra arguments.

A rule such as

```
p(X) --> q(X).
```

translates into

```
p(X, S0, S) :- q(X, S0, S).
```

If there is more than one non-terminal on the right-hand side, as in

```
p(X, Y) -->
        q(X),
        r(X, Y),
        s(Y).
```

then corresponding input and output arguments are identified, as in

```
p(X, Y, S0, S) :-
        q(X, S0, S1),
        r(X, Y, S1, S2),
        r(Y, S2, S).
```

Terminals are translated using the built-in predicate `'C'/3` (this predicate is not normally useful in itself; it has been given the name `'C'` simply to avoid using up a more useful name). Then, for instance

```
p(X) --> [go,to], q(X), [stop].
```

is translated by

```
p(X, S0, S) :-
        'C'(S0, go, S1),
        'C'(S1, to, S2),
        q(X, S2, S3),
        'C'(S3, stop, S).
```

Extra conditions expressed as explicit procedure calls naturally translate as themselves, e.g.

```
p(X) --> [X], {integer(X), X>0}, q(X).
```

translates to

```
p(X, S0, S) :-
        'C'(S0, X, S1),
        integer(X),
        X>0,
        q(X, S1, S).
```

Similarly, a cut is translated literally.

Terminals on the left-hand side of a rule translate into an explicit list in the output argument of the main non-terminal, e.g.

```
is(N), [not] --> [aint].
```

becomes

```
is(N, S0, [not|S]) :- 'C'(S0, aint, S).
```

Disjunction has a fairly obvious translation, e.g.

```
args(X, Y) -->
        (   dir(X), [to], indir(Y)
        ;   indir(Y), dir(X)
        ).
```

translates to

```
args(X, Y, S0, S) :-
        (   dir(X, S0, S1),
            'C'(S1, to, S2),
            indir(Y, S2, S)
        ;   indir(Y, S0, S1),
            dir(X, S1, S)
        ).
```

41.1 Usage and interface (dcg_doc)

- **Library usage:**
  `:- use_package(dcg).`
  or
  `:- module(...,...,[dcg]).`

42 Definite Clause Grammars (expansion)

Author(s): Daniel Cabeza.

This module implements the translation for Definite Clause Grammars (DCGs) (expansion).

42.1 Usage and interface (dcg_tr)

- **Library usage:**
  `:- use_module(library(dcg_tr)).`
- **Exports:**
  - *Predicates:*
    `phrase/2`, `phrase/3`, `dcg_translation/2`.
- **Other modules used:**
  - *System library modules:*
    `terms`, `assertions/doc_props`.

42.2 Documentation on exports (dcg_tr)

phrase/2: PREDICATE

`phrase(Phrase,List)`

Like `phrase(Phrase,List,[])`.

Meta-predicate with arguments: `phrase(goal,?)`.

phrase/3: PREDICATE

Meta-predicate with arguments: `phrase(goal,?,?)`.

Usage: `phrase(Phrase,List,Remainder)`

- *Description:* The list `List` is a phrase of type `Phrase` (according to the current grammar rules), where `Phrase` is either a non-terminal or more generally a grammar rule body. `Remainder` is what remains of the list after a phrase has been found.
- *The following properties should hold at call time:*
  `Phrase` is currently a term which is not a free variable. (term_typing:nonvar/1)
- *The following properties should hold globally:*
  Documentation is still incomplete: `phrase(Phrase,List,Remainder)` may not conform the functionality documented. (doc_props:doc_incomplete/1)

dcg_translation/2: PREDICATE

Performs the code expansion of source clauses that use DCGs.

42.3 Known bugs and planned improvements (dcg_tr)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

43 Formatted output

Author(s): The CLIP Group.

The `format` family of predicates is due to Quintus Prolog. They act as a Prolog interface to the C `stdio` function `printf()`, allowing formatted output.

Output is formatted according to an output pattern which can have either a format control sequence or any other character, which will appear verbatim in the output. Control sequences act as place-holders for the actual terms that will be output. Thus

```
?- format("Hello ~q!",world).
```

will print `Hello world!`.

If there is only one item to print it may be supplied alone. If there are more they have to be given as a list. If there are none then an empty list should be supplied. There has to be as many items as control characters.

The character `~` introduces a control sequence. To print a `~` verbatim just repeat it:

```
?- format("Hello ~~world!", []).
```

will result in `Hello ~world!`.

A format may be spread over several lines. The control sequence `\c` followed by a ⟨LFD⟩ will translate to the empty string:

```
?- format("Hello \c
world!", []).
```

will result in `Hello world!`.

43.1 Usage and interface (`format`)

- **Library usage:**
  `:- use_module(library(format)).`
- **Exports:**
  - *Predicates:*
    `format/2`, `format/3`, `sformat/3`, `format_to_string/3`.
  - *Regular Types:*
    `format_control/1`.
- **Other modules used:**
  - *System library modules:*
    `write`, `system`, `assertions/doc_props`.

43.2 Documentation on exports (`format`)

format/2: PREDICATE

General properties: `format(C,A)`

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP as `format(C,A)`. (basic_props:native/2)

Usage: `format(Format,Arguments)`

– *Description:* Print `Arguments` onto current output stream according to format `Format`.

– *The following properties should hold at call time:*

`Format` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

format/3: PREDICATE

General properties: `format(S,C,A)`

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP as `format(S,C,A)`. (basic_props:native/2)

Usage: `format(Stream,Format,Arguments)`

– *Description:* Print `Arguments` onto `Stream` according to format `Format`.

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

`Format` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

sformat/3: PREDICATE

Usage: `sformat(String,Format,Arguments)`

– *Description:* Same as `format_to_string(Format, Arguments, String)` (note the different argument order).

– *The following properties should hold at call time:*

`Format` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

– *The following properties should hold upon exit:*

`String` is a string (a list of character codes). (basic_props:string/1)

format_to_string/3: PREDICATE

Usage: `format_to_string(Format,Arguments,String)`

– *Description:* Print `Arguments` onto current string `String` according to format `Format`. This predicate is similar to the format/2, but the result is stored in a string.

– *The following properties should hold at call time:*

`Format` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

`Arguments` is a list. (basic_props:list/1)

– *The following properties should hold upon exit:*

`String` is a string (a list of character codes). (basic_props:string/1)

format_control/1: REGTYPE

The general format of a control sequence is `~NC`. The character `C` determines the type of the control sequence. `N` is an optional numeric argument. An alternative form of `N` is `*`. `*` implies that the next argument in `Arguments` should be used as a numeric argument in the control sequence. Example:

```
?- format("Hello~4cworld!", [0'x]).
```

and

```
?- format("Hello~*cworld!", [4,0'x]).
```

both produce

```
Helloxxxxworld!
```

The following control sequences are available.

- `~a` The argument is an atom. The atom is printed without quoting.
- `~Nc` (Print character.) The argument is a number that will be interpreted as an ASCII code. `N` defaults to one and is interpreted as the number of times to print the character.
- `~Ne`
- `~NE`
- `~Nf`
- `~Ng`
- `~NG` (Print float). The argument is a float. The float and `N` will be passed to the C `printf()` function as

  ```
  printf("%.Ne", Arg)
  printf("%.NE", Arg)
  printf("%.Nf", Arg)
  printf("%.Ng", Arg)
  printf("%.NG", Arg)
  ```

  If `N` is not supplied the action defaults to

  ```
  printf("%e", Arg)
  printf("%E", Arg)
  printf("%f", Arg)
  printf("%g", Arg)
  printf("%G", Arg)
  ```

- `~Nd` (Print decimal.) The argument is an integer. `N` is interpreted as the number of digits after the decimal point. If `N` is 0 or missing, no decimal point will be printed. Example:

  ```
  ?- format("Hello ~1d world!", [42]).
  ?- format("Hello ~d world!", [42]).
  ```

  will print as

  ```
  Hello 4.2 world!
  Hello 42 world!
  ```

  respectively.
- `~ND` (Print decimal.) The argument is an integer. Identical to `~Nd` except that `,` will separate groups of three digits to the left of the decimal point. Example:

  ```
  ?- format("Hello ~1D world!", [12345]).
  ```

  will print as

  ```
  Hello 1,234.5 world!
  ```

- ~Nr (Print radix.) The argument is an integer. N is interpreted as a radix. N should be >= 2 and <= 36. If N is missing the radix defaults to 8. The letters a-z will denote digits larger than 9. Example:

```
?- format("Hello ~2r world!", [15]).
?- format("Hello ~16r world!", [15]).
```

  will print as

```
Hello 1111 world!
Hello f world!
```

  respectively.
- ~NR (Print radix.) The argument is an integer. Identical to ~Nr except that the letters A-Z will denote digits larger than 9. Example:

```
?- format("Hello ~16R world!", [15]).
```

  will print as

```
Hello F world!
```

- ~Ns (Print string.) The argument is a list of ASCII codes. Exactly N characters will be printed. N defaults to the length of the string. Example:

```
?- format("Hello ~4s ~4s!", ["new","world"]).
?- format("Hello ~s world!", ["new"]).
```

  will print as

```
Hello new  worl!
Hello new world!
```

  respectively.
- ~i (Ignore argument.) The argument may be of any type. The argument will be ignored. Example:

```
?- format("Hello ~i~s world!", ["old","new"]).
```

  will print as

```
Hello new world!
```

- ~k (Print canonical.) The argument may be of any type. The argument will be passed to write_canonical/2 (Chapter 36 [Term output], page 249). Example:

```
?- format("Hello ~k world!", [[a,b,c]]).
```

  will print as

```
Hello .(a,.(b,.(c,[]))) world!
```

- ~p (print.) The argument may be of any type. The argument will be passed to print/2 (Chapter 36 [Term output], page 249). Example:

  suposing the user has defined the predicate

```
:- multifile portray/1.
portray([X|Y]) :- print(cons(X,Y)).
```

  then

```
?- format("Hello ~p world!", [[a,b,c]]).
```

  will print as

```
Hello cons(a,cons(b,cons(c,[]))) world!
```

- ~q (Print quoted.) The argument may be of any type. The argument will be passed to writeq/2 (Chapter 36 [Term output], page 249). Example:

```
?- format("Hello ~q world!", [['A','B']]).
```

  will print as

```
Hello ['A','B'] world!
```

- ~w (write.) The argument may be of any type. The argument will be passed to `write/2` (Chapter 36 [Term output], page 249). Example:

  ```
  ?- format("Hello ~w world!", [['A','B']]).
  ```

  will print as

  ```
  Hello [A,B] world!
  ```

- ~Nn (Print newline.) Print `N` newlines. `N` defaults to 1. Example:

  ```
  ?- format("Hello ~n world!", []).
  ```

  will print as

  ```
  Hello
   world!
  ```

- ~N (Fresh line.) Print a newline, if not already at the beginning of a line.
- ~~ (Print tilde.) Prints ~

The following control sequences are also available for compatibility, but do not perform any useful functions.

- ~N| (Set tab.) Set a tab stop at position `N`, where `N` defaults to the current position, and advance the current position there.
- ~N+ (Advance tab.) Set a tab stop at `N` positions past the current position, where `N` defaults to 8, and advance the current position there.
- ~Nt (Set fill character.) Set the fill character to be used in the next position movement to `N`, where `N` defaults to ⟨SPC⟩.

Usage: `format_control(C)`

– *Description:* `C` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`.

– *The following properties should hold globally:*

Documentation is still incomplete: `format_control(C)` may not conform the functionality documented. (doc_props:doc_incomplete/1)

43.3 Known bugs and planned improvements (`format`)

- `format_to_string/3` is extremelly slow in its current implementation. It writes an intermediate file which is immediately removed, if possible. It is not very fast. In case there are no permissions, writing is attempted through an internal pipe, which may hang if the string is too long (this is O.S. dependant).

44 List processing

Author(s): The CLIP Group.
This module provides a set of predicates for list processing.

44.1 Usage and interface (`lists`)

- **Library usage:**
  `:- use_module(library(lists)).`
- **Exports:**
  - *Predicates:*
    `nonsingle/1`, `append/3`, `reverse/2`, `reverse/3`, `delete/3`, `delete_non_ground/3`, `select/3`, `length/2`, `nth/3`, `add_after/4`, `add_before/4`, `dlist/3`, `list_concat/2`, `list_insert/2`, `insert_last/3`, `contains_ro/2`, `contains1/2`, `nocontainsx/2`, `last/2`, `list_lookup/3`, `list_lookup/4`, `intset_insert/3`, `intset_delete/3`, `intset_in/2`, `intset_sequence/3`, `intersection/3`, `union/3`, `difference/3`, `equal_lists/2`, `list_to_list_of_lists/2`, `powerset/2`, `cross_product/2`, `sequence_to_list/2`.
  - *Properties:*
    `list1/2`, `sublist/2`, `subordlist/2`.
  - *Regular Types:*
    `list_of_lists/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

44.2 Documentation on exports (`lists`)

nonsingle/1: PREDICATE

General properties: `nonsingle(A)`

- *Description:* nonsingle fails.
- *If the following properties should hold at call time:*
  term_basic:A=[a] (term_basic:= /2)
  then the following properties should hold globally:
  Calls of the form `nonsingle(A)` fail. (native_props:fails/1)

`nonsingle(A)`

- *Description:* nonsingle succeeds.
- *If the following properties should hold at call time:*
  term_basic:A=[a,b] (term_basic:= /2)

Usage: `nonsingle(X)`

- *Description:* `X` is not a singleton.

append/3: PREDICATE

General properties: `append(Xs,Ys,Zs)`

– *Description:* `Zs` is `Ys` appended to `Xs`.

– *The following properties should hold upon exit:*

`Xs` is a list. (basic_props:list/1)

`append(Xs,Ys,Zs)`

– *If the following properties hold at call time:*

`Xs` is a list. (basic_props:list/1)

`Ys` is a list. (basic_props:list/1)

then the following properties should hold upon exit:

`Zs` is a list. (basic_props:list/1)

`append(Xs,Ys,Zs)`

– *If the following properties hold at call time:*

`Zs` is a list. (basic_props:list/1)

then the following properties should hold upon exit:

`Xs` is a list. (basic_props:list/1)

`Ys` is a list. (basic_props:list/1)

`append(Xs,Ys,Zs)`

– *The following properties should hold upon exit:*

The sharing pattern is `[[X,Y,Z],[X,Z],[Y,Z]]`. (native_props:mshare/1)

`append(Xs,Ys,Zs)`

– *If the following properties hold at call time:*

`Xs` is currently ground (it contains no variables). (term_typing:ground/1)

`Ys` is currently ground (it contains no variables). (term_typing:ground/1)

then the following properties should hold upon exit:

`Zs` is currently ground (it contains no variables). (term_typing:ground/1)

`append(Xs,Ys,Zs)`

– *If the following properties hold at call time:*

`Zs` is currently ground (it contains no variables). (term_typing:ground/1)

then the following properties should hold upon exit:

`Xs` is currently ground (it contains no variables). (term_typing:ground/1)

`Ys` is currently ground (it contains no variables). (term_typing:ground/1)

`append(Xs,Ys,Zs)`

– *The following properties hold globally:*

`append(Xs,Ys,Zs)` is side-effect `free`. (basic_props:sideff/2)

`append(Xs,Ys,Zs)`

– *If the following properties hold at call time:*

`Xs` is a list. (basic_props:list/1)

then the following properties hold globally:

`append(Xs,Ys,Zs)` is evaluable at compile-time. (basic_props:eval/1)

`append(Xs,Ys,Zs)`

– *If the following properties hold at call time:*
`Zs` is a list. (basic_props:list/1)
then the following properties hold globally:
`append(Xs,Ys,Zs)` is evaluable at compile-time. (basic_props:eval/1)

`append(A,B,C)`
– *Description:* Simple call to append
– *If the following properties should hold at call time:*
term_basic:A=[1,2,3] (term_basic:= /2)
term_basic:B=[4,5] (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:C=[1,2,3,4,5] (term_basic:= /2)

`append(A,B,X)`
– *Description:* Simple append test
– *If the following properties should hold at call time:*
term_basic:L=[([1,2],[1,2,4,5,6]),([1,2,3],[1,2,3,4,5,6])] (term_basic:= /2)
term_basic:B=[4,5,6] (term_basic:= /2)
`A,X2` is an element of `L`. (basic_props:member/2)
then the following properties should hold upon exit:
The terms `X` and `X2` are strictly identical. (term_compare:== /2)

`append(A,B,X)`
– *Description:* Test of reverse call
– *If the following properties should hold at call time:*
term_basic:X=[1,2,3] (term_basic:= /2)
then the following properties should hold upon exit:
`A,B` is an element of `[([],[1,2,3]),([1],[2,3]),([1,2],[3]),([1,2,3],[])]`. (basic_props:member/2)

`append(A,B,X)`
– *Description:* Empty test.
– *If the following properties should hold at call time:*
term_basic:A=[] (term_basic:= /2)
term_basic:B=[] (term_basic:= /2)
then the following properties should hold upon exit:
The terms `X` and `[]` are strictly identical. (term_compare:== /2)

`append(_A,B,X)`
– *Description:* Test of a call that fails
– *If the following properties should hold at call time:*
term_basic:B=[2] (term_basic:= /2)
term_basic:X=[1,2,3] (term_basic:= /2)
then the following properties should hold globally:
Calls of the form `append(_A,B,X)` fail. (native_props:fails/1)

`append(X,Y,Z)`
– *Description:* Test of a reverse call.

– *If the following properties should hold at call time:*
term_basic:Y=[2] (term_basic:= /2)
term_basic:Z=[1,2] (term_basic:= /2)
then the following properties should hold upon exit:
The terms `X` and `[1]` are strictly identical. (term_compare:== /2)

Usage: `append(Xs,Ys,Zs)`

– *Call and exit should be compatible with:*
`Xs` is a list. (basic_props:list/1)
`Ys` is a list. (basic_props:list/1)
`Zs` is a list. (basic_props:list/1)

reverse/2: PREDICATE

General properties: `reverse(A,B)`

– *Description:* Reverse a list
– *If the following properties should hold at call time:*
term_basic:A=[1,2,3] (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:B=[3,2,1] (term_basic:= /2)
– *The following properties hold globally:*
`reverse(Arg1,Arg2)` is side-effect `free`. (basic_props:sideff/2)

`reverse(Xs,_Ys)`

– *If the following properties hold at call time:*
`Xs` is a list. (basic_props:list/1)
then the following properties hold globally:
`reverse(Xs,_Ys)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `reverse(Xs,Ys)`

– *Description:* Reverses the order of elements in `Xs`.
– *The following properties should hold at call time:*
`Xs` is a list. (basic_props:list/1)
`Ys` is any term. (basic_props:term/1)
– *The following properties should hold upon exit:*
`Xs` is a list. (basic_props:list/1)
`Ys` is a list. (basic_props:list/1)

reverse/3: PREDICATE

General properties: `reverse(A,B,C)`

– *Description:* reverse/3 test
– *Call and exit should be compatible with:*
`B` is a free variable. (term_typing:var/1)
– *If the following properties should hold at call time:*
term_basic:A=[1,2,3] (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:C=[3,2,1|B] (term_basic:= /2)

– *The following properties hold globally:*
`reverse(Arg1,Arg2,Arg3)` is side-effect `free`. (basic_props:sideff/2)

`reverse(Xs,Ys,Zs)`
– *If the following properties hold at call time:*
`Xs` is a list. (basic_props:list/1)
`Ys` is a list. (basic_props:list/1)
then the following properties hold globally:
`reverse(Xs,Ys,Zs)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `reverse(A,B,C)`
– *Description:* Reverse the order of elements in `A`, and append it with `B`.

delete/3: PREDICATE

General properties:
– *The following properties hold globally:*
`delete(Arg1,Arg2,Arg3)` is side-effect `free`. (basic_props:sideff/2)

`delete(L1,E,L2)`
– *If the following properties hold at call time:*
`L1` is currently ground (it contains no variables). (term_typing:ground/1)
`L2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`delete(L1,E,L2)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `delete(L1,E,L2)`
– *Description:* `L2` is `L1` without the ocurrences of `E`.
– *The following properties should hold upon exit:*
`L1` is a list. (basic_props:list/1)
`L2` is a list. (basic_props:list/1)

delete_non_ground/3: PREDICATE

General properties:
– *The following properties hold globally:*
`delete_non_ground(Arg1,Arg2,Arg3)` is side-effect `true`. (basic_props:sideff/2)

`delete_non_ground(L1,E,L2)`
– *If the following properties hold at call time:*
`L1` is currently ground (it contains no variables). (term_typing:ground/1)
`L2` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`delete_non_ground(L1,E,L2)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `delete_non_ground(L1,E,L2)`
– *Description:* `L2` is `L1` without the ocurrences of `E`. `E` can be a nonground term so that all the elements in `L1` it unifies with will be deleted
– *The following properties should hold upon exit:*
`L1` is a list. (basic_props:list/1)
`L2` is a list. (basic_props:list/1)

select/3: PREDICATE

General properties:

– *The following properties hold globally:*
`select(Arg1,Arg2,Arg3)` is side-effect `free`. (basic_props:sideff/2)

`select(X,Xs,Ys)`

– *If the following properties hold at call time:*
`X` is currently ground (it contains no variables). (term_typing:ground/1)
`Xs` is currently ground (it contains no variables). (term_typing:ground/1)
then the following properties hold globally:
`select(X,Xs,Ys)` is evaluable at compile-time. (basic_props:eval/1)

Usage: `select(X,Xs,Ys)`

– *Description:* `Xs` and `Ys` have the same elements except for one occurrence of `X`.

length/2: PREDICATE

General properties: `length(A,B)`

– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)
– *The following properties hold globally:*
`length(Arg1,Arg2)` is side-effect `free`. (basic_props:sideff/2)

`length(L,N)`

– *If the following properties hold at call time:*
`L` is a list. (basic_props:list/1)
then the following properties hold globally:
`length(L,N)` is evaluable at compile-time. (basic_props:eval/1)

`length(L,N)`

– *If the following properties hold at call time:*
`N` is currently instantiated to an integer. (term_typing:integer/1)
then the following properties hold globally:
`length(L,N)` is evaluable at compile-time. (basic_props:eval/1)

Usage 1: `length(L,N)`

– *Description:* Computes the length of `L`.
– *The following properties should hold at call time:*
`L` is a list. (basic_props:list/1)
`N` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`L` is a list. (basic_props:list/1)
`N` is an integer. (basic_props:int/1)

Usage 2: `length(L,N)`

– *Description:* Outputs `L` of length `N`.
– *The following properties should hold at call time:*
`L` is a free variable. (term_typing:var/1)
`N` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*
`L` is a list. (basic_props:list/1)
`N` is an integer. (basic_props:int/1)

Usage 3: `length(L,N)`
– *Description:* Checks that `L` is of length `N`.
– *The following properties should hold at call time:*
`L` is a list. (basic_props:list/1)
`N` is an integer. (basic_props:int/1)
– *The following properties should hold upon exit:*
`L` is a list. (basic_props:list/1)
`N` is an integer. (basic_props:int/1)

nth/3: PREDICATE

`nth(N,List,Elem)`
`N` is the position in `List` of `Elem`. `N` counts from one.
General properties:
– *The following properties hold globally:*
`nth(N,List,Elem)` is side-effect `free`. (basic_props:sideff/2)

`nth(N,L,E)`
– *If the following properties hold at call time:*
`N` is currently instantiated to an integer. (term_typing:integer/1)
then the following properties hold globally:
`nth(N,L,E)` is evaluable at compile-time. (basic_props:eval/1)

`nth(N,L,E)`
– *If the following properties hold at call time:*
`L` is a list. (basic_props:list/1)
then the following properties hold globally:
`nth(N,L,E)` is evaluable at compile-time. (basic_props:eval/1)

`nth(N,L,E)`
– *If the following properties hold at call time:*
`N` is an integer. (basic_props:int/1)
`L` is a list. (basic_props:list/1)
then the following properties hold globally:
All calls of the form `nth(N,L,E)` are deterministic. (native_props:is_det/1)

Usage 1:
– *Description:* Unifies `Elem` and the `N`th element of `List`.
– *Call and exit should be compatible with:*
`List` is a list. (basic_props:list/1)
`Elem` is any term. (basic_props:term/1)
– *The following properties should hold at call time:*
`N` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`List` is a list. (basic_props:list/1)

`Elem` is any term. (basic_props:term/1)

Usage 2:

– *Description:* Finds the positions where `Elem` is in `List`. Positions are found in ascending order.

– *Call and exit should be compatible with:*

`List` is a list. (basic_props:list/1)

`Elem` is any term. (basic_props:term/1)

– *The following properties should hold at call time:*

`N` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`N` is an integer. (basic_props:int/1)

`List` is a list. (basic_props:list/1)

`Elem` is any term. (basic_props:term/1)

add_after/4: PREDICATE

Usage: `add_after(L0,E0,E,L)`

– *Description:* Adds element `E` after element `E0` (or at end) to list `L0` returning in `L` the new list (uses term comparison).

– *The following properties should hold at call time:*

`L0` is currently a term which is not a free variable. (term_typing:nonvar/1)

`E0` is currently a term which is not a free variable. (term_typing:nonvar/1)

`E` is currently a term which is not a free variable. (term_typing:nonvar/1)

`L` is a free variable. (term_typing:var/1)

add_before/4: PREDICATE

Usage: `add_before(L0,E0,E,L)`

– *Description:* Adds element E before element E0 (or at start) to list L0 returning in L the new list (uses term comparison).

– *The following properties should hold at call time:*

`L0` is currently a term which is not a free variable. (term_typing:nonvar/1)

`E0` is currently a term which is not a free variable. (term_typing:nonvar/1)

`E` is currently a term which is not a free variable. (term_typing:nonvar/1)

`L` is a free variable. (term_typing:var/1)

list1/2: PROPERTY

Meta-predicate with arguments: `list1(?,(pred 1))`.

Usage: `list1(X,Y)`

– *Description:* `X` is a list of `Y`s of at least one element.

dlist/3: PREDICATE

Usage: `dlist(List,DList,Tail)`

– *Description:* `List` is the result of removing `Tail` from the end of `DList` (makes a difference list from a list).

list_concat/2: PREDICATE

Usage: `list_concat(LL,L)`

– *Description:* `L` is the concatenation of all the lists in `LL`.

– *The following properties should hold at call time:*

`LL` is a list of `lists`. (basic_props:list/2)

– *The following properties should hold upon exit:*

`L` is a list. (basic_props:list/1)

list_insert/2: PREDICATE

Usage: `list_insert(List,Term)`

– *Description:* Adds `Term` to the end of `List` if there is no element in `List` identical to `Term`.

– *The following properties should hold at call time:*

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

insert_last/3: PREDICATE

Usage: `insert_last(L0,E,L)`

– *Description:* Adds element `E` at end of list `L0` returning `L`.

– *The following properties should hold at call time:*

`L0` is currently a term which is not a free variable. (term_typing:nonvar/1)

`E` is currently a term which is not a free variable. (term_typing:nonvar/1)

contains_ro/2: PREDICATE

Usage:

– *Description:* Impure membership (does not instantiate a variable in its first argument.

contains1/2: PREDICATE

Usage:

– *Description:* First membership.

nocontainsx/2: PREDICATE

Usage: `nocontainsx(L,X)`

– *Description:* `X` is not identical to any element of `L`.

last/2: PREDICATE

Usage: `last(L,X)`

– *Description:* `X` is the last element of list `L`.

list_lookup/3: PREDICATE

Usage: `list_lookup(List,Key,Value)`

– *Description:* Same as `list_lookup/4`, but use `-/2` as functor.

list_lookup/4: PREDICATE

Usage: `list_lookup(List,Functor,Key,Value)`

– *Description:* Look up `Functor(Key,Value)` pair in variable ended key-value pair list `L` or else add it at the end.

intset_insert/3: PREDICATE

Usage: `intset_insert(A,B,Set)`

– *Description:* Insert the element `B` in the ordered set of numbers `A`.

intset_delete/3: PREDICATE

Usage: `intset_delete(A,B,Set)`

– *Description:* Delete from the ordered set `A` the element `B`.

intset_in/2: PREDICATE

Usage: `intset_in(E,Set)`

– *Description:* Succeds iff `E` is element of `Set`

intset_sequence/3: PREDICATE

Usage: `intset_sequence(N,L1,L2)`

– *Description:* Generates an ordered set of numbers from 0 to `N`-1, and append it to `L1`.

intersection/3: PREDICATE

Usage: `intersection(List1,List2,List)`

– *Description:* `List` has the elements which are both in `List1` and `List2`.

– *The following properties should hold at call time:*

`List1` is currently a term which is not a free variable. (term_typing:nonvar/1)

`List2` is currently a term which is not a free variable. (term_typing:nonvar/1)

`List1` is a list. (basic_props:list/1)

`List2` is a list. (basic_props:list/1)

– *The following properties should hold upon exit:*

`List` is a list. (basic_props:list/1)

union/3: PREDICATE

Usage: `union(List1,List2,List)`

- *Description:* `List` has the elements which are in `List1` followed by the elements which are in `List2` but not in `List1`.
- *The following properties should hold at call time:*

  `List1` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `List2` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `List` is a free variable. (term_typing:var/1)
  `List1` is a list. (basic_props:list/1)
  `List2` is a list. (basic_props:list/1)
- *The following properties should hold upon exit:*

  `List` is a list. (basic_props:list/1)

difference/3: PREDICATE

Usage: `difference(List1,List2,List)`

- *Description:* `List` has the elements which are in `List1` but not in `List2`.
- *The following properties should hold at call time:*

  `List1` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `List2` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `List` is a free variable. (term_typing:var/1)
  `List1` is a list. (basic_props:list/1)
  `List2` is a list. (basic_props:list/1)
- *The following properties should hold upon exit:*

  `List` is a list. (basic_props:list/1)

sublist/2: PROPERTY

Usage: `sublist(List1,List2)`

- *Description:* `List2` contains all the elements of `List1`.
- *If the following properties should hold at call time:*

  `List2` is currently a term which is not a free variable. (term_typing:nonvar/1)

subordlist/2: PROPERTY

Usage: `subordlist(List1,List2)`

- *Description:* `List2` contains all the elements of `List1` in the same order.
- *If the following properties should hold at call time:*

  `List2` is currently a term which is not a free variable. (term_typing:nonvar/1)

equal_lists/2: PREDICATE

Usage: `equal_lists(List1,List2)`

- *Description:* `List1` has all the elements of `List2`, and vice versa.
- *The following properties should hold at call time:*

  `List1` is a list. (basic_props:list/1)
  `List2` is a list. (basic_props:list/1)

list_to_list_of_lists/2: PREDICATE

Usage 1: `list_to_list_of_lists(List,LList)`

- *The following properties should hold at call time:*
  `List` is a list. (basic_props:list/1)
- *The following properties should hold upon exit:*
  lists:list_of_lists(LList) (lists:list_of_lists/1)

Usage 2: `list_to_list_of_lists(List,LList)`

- *Description:* `LList` is the list of one element lists with elements of `List`.
- *The following properties should hold at call time:*
  lists:list_of_lists(LList) (lists:list_of_lists/1)
- *The following properties should hold upon exit:*
  `List` is a list. (basic_props:list/1)

powerset/2: PREDICATE

Usage: `powerset(List,LList)`

- *Description:* `LList` is the powerset of `List`, i.e., the list of all lists which have elements of `List`. If `List` is ordered, `LList` and all its elements are ordered.
- *The following properties should hold at call time:*
  `List` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `List` is a list. (basic_props:list/1)
- *The following properties should hold upon exit:*
  lists:list_of_lists(LList) (lists:list_of_lists/1)

cross_product/2: PREDICATE

Usage: `cross_product(LList,List)`

- *Description:* `List` is the cartesian product of the lists in `LList`, that is, the list of lists formed with one element of each list in `LList`, in the same order.
- *The following properties should hold at call time:*
  `LList` is currently a term which is not a free variable. (term_typing:nonvar/1)
  lists:list_of_lists(LList) (lists:list_of_lists/1)
- *The following properties should hold upon exit:*
  lists:list_of_lists(List) (lists:list_of_lists/1)

sequence_to_list/2: PREDICATE

Usage: `sequence_to_list(Sequence,List)`

- *Description:* `List` is the list of all elements in the (comma-separated) sequence `Sequence`. The use of this predicate is reversible.

list_of_lists/1: REGTYPE

A regular type, defined as follows:

```
list_of_lists([]).
list_of_lists([L|Xs]) :-
        list(L),
        list_of_lists(Xs).
```

45 Sorting lists

Author(s): Richard A. O'Keefe (original version), The CLIP Group (changes and modifications).

This module implements some sorting list predicates.

45.1 Usage and interface (`sort`)

- **Library usage:**
  `:- use_module(library(sort)).`
- **Exports:**
  - *Predicates:*
    `sort/2`, `keysort/2`.
  - *Regular Types:*
    `keylist/1`, `keypair/1`.

45.2 Documentation on exports (`sort`)

sort/2: PREDICATE

`sort(List1,List2)`

The elements of `List1` are sorted into the standard order (see Chapter 18 [Comparing terms], page 135) and any identical elements are merged, yielding `List2`. The time and space complexity of this operation is at worst `O(N lg N)` where `N` is the length of `List1`.

General properties: `sort(A,B)`

- *If the following properties should hold at call time:*
  term_basic:A=[1,2,6,5,2,1] (term_basic:= /2)
  then the following properties should hold upon exit:
  The terms `B` and `[1,2,5,6]` are strictly identical. (term_compare:== /2)

`sort(A,B)`

- *If the following properties hold at call time:*
  `A` is a list. (basic_props:list/1)
  then the following properties hold globally:
  `sort(A,B)` is evaluable at compile-time. (basic_props:eval/1)

`sort(A,B)`

- *The following properties hold globally:*
  `sort(A,B)` is side-effect `free`. (basic_props:sideff/2)

Usage:

- *Description:* `List2` is the sorted list corresponding to `List1`.
- *Call and exit should be compatible with:*
  `List2` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*
  `List1` is a list. (basic_props:list/1)

– *The following properties should hold upon exit:*
`List2` is a list. (basic_props:list/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

keysort/2: PREDICATE

`keysort(List1,List2)`

`List1` is sorted into order according to the value of the *keys* of its elements, yielding the list `List2`. No merging takes place. This predicate is *stable*, i.e., if an element `A` occurs before another element `B` *with the same key* in the input, then `A` will occur before `B` also in the output. The time and space complexity of this operation is at worst `O(N lg N)` where `N` is the length of `List1`.

Usage:

– *Description:* `List2` is the (key-)sorted list corresponding to `List1`.
– *Call and exit should be compatible with:*
`List2` is a list of pairs of the form `Key-Value`. (sort:keylist/1)
– *The following properties should hold at call time:*
`List1` is a list of pairs of the form `Key-Value`. (sort:keylist/1)
– *The following properties should hold upon exit:*
`List2` is a list of pairs of the form `Key-Value`. (sort:keylist/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

keylist/1: REGTYPE

Usage: `keylist(L)`

– *Description:* `L` is a list of pairs of the form `Key-Value`.

keypair/1: REGTYPE

Usage: `keypair(P)`

– *Description:* `P` is a pair of the form `"K-_"`, where `K` is considered the *key*.

45.3 Known bugs and planned improvements (`sort`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

46 compiler (library)

46.1 Usage and interface (compiler)

- **Library usage:**
  `:- use_module(library(compiler)).`
- **Exports:**
  - *Predicates:*
    `make_po/1`, `make_wam/1`, `ensure_loaded/1`, `ensure_loaded/2`, `use_module/1`, `use_module/2`, `use_module/3`, `unload/1`, `set_debug_mode/1`, `set_nodebug_mode/1`, `set_debug_module/1`, `set_nodebug_module/1`, `set_debug_module_source/1`, `mode_of_module/2`, `module_of/2`.
- **Other modules used:**
  - *System library modules:*
    `system`, `compiler/c_itf_internal`, `compiler/exemaker`.

46.2 Documentation on exports (compiler)

make_po/1: PREDICATE
No further documentation available for this predicate.

make_wam/1: PREDICATE
No further documentation available for this predicate.

ensure_loaded/1: PREDICATE
No further documentation available for this predicate.

ensure_loaded/2: PREDICATE
No further documentation available for this predicate.

use_module/1: PREDICATE
No further documentation available for this predicate.

use_module/2: PREDICATE
No further documentation available for this predicate.
Meta-predicate with arguments: `use_module(?,addmodule(?))`.

use_module/3: PREDICATE
No further documentation available for this predicate.

unload/1: PREDICATE
No further documentation available for this predicate.

set_debug_mode/1: PREDICATE
No further documentation available for this predicate.

set_nodebug_mode/1: PREDICATE
No further documentation available for this predicate.

set_debug_module/1: PREDICATE
No further documentation available for this predicate.

set_nodebug_module/1: PREDICATE
No further documentation available for this predicate.

set_debug_module_source/1: PREDICATE
No further documentation available for this predicate.

mode_of_module/2: PREDICATE
No further documentation available for this predicate.

module_of/2: PREDICATE
No further documentation available for this predicate.

47 Enumeration of integers inside a range

Author(s): The CLIP Group.

This modules enumerates integers between two numbers, or checks that an integer lies within a range

47.1 Usage and interface (`between`)

- **Library usage:**
  `:- use_module(library(between)).`
- **Exports:**
  - *Predicates:*
    `between/3`.

47.2 Documentation on exports (`between`)

between/3: PREDICATE

Usage: `between(Min,Max,N)`

- *Description:* `N` is a number which is greater than or equal to `Min` and smaller than or equal to `Max`. Both `Min` and `Max` can be either integer or real numbers.
- *The following properties should hold at call time:*
  `Min` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Max` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Min` is a number. (basic_props:num/1)
  `Max` is a number. (basic_props:num/1)
- *The following properties should hold upon exit:*
  `N` is an integer. (basic_props:int/1)

48 Operating system utilities

Author(s): Daniel Cabeza, Manuel Carro.

This module contains predicates for invoking services which are typically provided by the operating system. Note that the predicates which take names of files or directories as arguments in this module expect atoms, not path aliases. I.e., generally these predicates will not call `absolute_file_name/2` on names of files or directories taken as arguments.

48.1 Usage and interface (`system`)

- **Library usage:**
  `:- use_module(library(system)).`
- **Exports:**
  - *Predicates:*
    `pause/1`, `time/1`, `datime/1`, `datime/9`, `getenvstr/2`, `setenvstr/2`, `current_env/2`, `set_env/2`, `del_env/1`, `c_errno/1`, `copy_file/2`, `copy_file/3`, `dir_path/2`, `extract_paths/2`, `file_dir_name/3`, `get_pid/1`, `get_uid/1`, `get_gid/1`, `get_pwnam/1`, `get_grnam/1`, `get_tmp_dir/1`, `get_address/2`, `current_host/1`, `current_executable/1`, `umask/2`, `make_directory/2`, `make_directory/1`, `make_dirpath/2`, `make_dirpath/1`, `working_directory/2`, `cd/1`, `shell/0`, `shell/1`, `shell/2`, `system/1`, `system/2`, `popen/3`, `exec/4`, `exec/3`, `exec/8`, `wait/3`, `directory_files/2`, `mktemp/2`, `mktemp_in_tmp/2`, `file_exists/1`, `file_exists/2`, `file_property/2`, `file_properties/6`, `modif_time/2`, `modif_time0/2`, `fmode/2`, `chmod/2`, `chmod/3`, `set_exec_mode/2`, `delete_file/1`, `delete_directory/1`, `rename_file/2`, `using_windows/0`, `winpath/2`, `winpath/3`, `winpath_c/3`, `cyg2win/3`, `no_swapslash/3`, `replace_characters/4`, `system_error_report/1`.
  - *Regular Types:*
    `datime_struct/1`, `popen_mode/1`.
  - *Multifiles:*
    `define_flag/3`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

48.2 Documentation on exports (`system`)

pause/1: PREDICATE

`pause(Seconds)`

Make this thread sleep for some `Seconds`.

Usage:

- *The following properties should hold at call time:*
  `Seconds` is an integer. (basic_props:int/1)

time/1: PREDICATE

`time(Time)`

`Time` is unified with the number of seconds elapsed since January, 1, 1970 (UTC).

Usage:

– *Calls should, and exit will be compatible with:*

`Time` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Time` is an integer. (basic_props:int/1)

datime/1: PREDICATE

`datime(Datime)`

`Datime` is unified with a term of the form `datime(Year,Month,Day,Hour,Minute,Second)` which contains the current date and time.

Usage:

– *Call and exit should be compatible with:*

system:datime_struct(Datime) (system:datime_struct/1)

– *The following properties should hold upon exit:*

system:datime_struct(Datime) (system:datime_struct/1)

datime/9: PREDICATE

`datime(Time,Year,Month,Day,Hour,Min,Sec,WeekDay,YearDay)`

`Time` is as in `time/1`. `WeekDay` is the number of days since Sunday, in the range 0 to 6. `YearDay` is the number of days since January 1, in the range 0 to 365.

Usage 1:

– *Description:* If `Time` is given, the rest of the arguments are unified with the date and time to which the `Time` argument refers.

– *Calls should, and exit will be compatible with:*

`Year` is an integer. (basic_props:int/1)

`Month` is an integer. (basic_props:int/1)

`Day` is an integer. (basic_props:int/1)

`Hour` is an integer. (basic_props:int/1)

`Min` is an integer. (basic_props:int/1)

`Sec` is an integer. (basic_props:int/1)

`WeekDay` is an integer. (basic_props:int/1)

`YearDay` is an integer. (basic_props:int/1)

– *The following properties should hold at call time:*

`Time` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Year` is an integer. (basic_props:int/1)

`Month` is an integer. (basic_props:int/1)

`Day` is an integer. (basic_props:int/1)

`Hour` is an integer. (basic_props:int/1)

`Min` is an integer. (basic_props:int/1)
`Sec` is an integer. (basic_props:int/1)
`WeekDay` is an integer. (basic_props:int/1)
`YearDay` is an integer. (basic_props:int/1)

Usage 2:

- *Description:* Bound `Time`, `WeekDay` and `YearDay` as determined by the input arguments.
- *Calls should, and exit will be compatible with:*
  `Time` is an integer. (basic_props:int/1)
  `WeekDay` is an integer. (basic_props:int/1)
  `YearDay` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*
  `Year` is an integer. (basic_props:int/1)
  `Month` is an integer. (basic_props:int/1)
  `Day` is an integer. (basic_props:int/1)
  `Hour` is an integer. (basic_props:int/1)
  `Min` is an integer. (basic_props:int/1)
  `Sec` is an integer. (basic_props:int/1)
- *The following properties hold upon exit:*
  `Time` is an integer. (basic_props:int/1)
  `WeekDay` is an integer. (basic_props:int/1)
  `YearDay` is an integer. (basic_props:int/1)

Usage 3:

- *Description:* Bound `Time` to current time and the rest of the arguments refer to current time.
- *Calls should, and exit will be compatible with:*
  `WeekDay` is an integer. (basic_props:int/1)
  `YearDay` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*
  `Time` is a free variable. (term_typing:var/1)
  `Year` is a free variable. (term_typing:var/1)
  `Month` is a free variable. (term_typing:var/1)
  `Day` is a free variable. (term_typing:var/1)
  `Hour` is a free variable. (term_typing:var/1)
  `Min` is a free variable. (term_typing:var/1)
  `Sec` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*
  `Time` is an integer. (basic_props:int/1)
  `Year` is an integer. (basic_props:int/1)
  `Month` is an integer. (basic_props:int/1)
  `Day` is an integer. (basic_props:int/1)
  `Hour` is an integer. (basic_props:int/1)
  `Min` is an integer. (basic_props:int/1)
  `Sec` is an integer. (basic_props:int/1)
  `WeekDay` is an integer. (basic_props:int/1)
  `YearDay` is an integer. (basic_props:int/1)

datime_struct/1: REGTYPE

A regular type, defined as follows:

```
datime_struct(datime(Year,Month,Day,Hour,Min,Sec)) :-
        int(Year),
        int(Month),
        int(Day),
        int(Hour),
        int(Min),
        int(Sec).
```

getenvstr/2: PREDICATE

`getenvstr(Name,Value)`

The environment variable `Name` has `Value`. Fails if variable `Name` is not defined.

Usage:

– *Call and exit should be compatible with:*

`Value` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold at call time:*

`Name` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*

`Value` is a string (a list of character codes). (basic_props:string/1)

setenvstr/2: PREDICATE

`setenvstr(Name,Value)`

The environment variable `Name` is assigned `Value`.

Usage:

– *The following properties should hold at call time:*

`Name` is an atom. (basic_props:atm/1)

`Value` is a string (a list of character codes). (basic_props:string/1)

current_env/2: PREDICATE

`current_env(Name,Value)`

If `Name` is an atom, then unifies the environment variable `Name` with its value. Note that this predicate can be used to enumerate all the environment variables using backtracking.

Usage:

– *Call and exit should be compatible with:*

`Name` is an atom. (basic_props:atm/1)

`Value` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*

`Name` is an atom. (basic_props:atm/1)

`Value` is an atom. (basic_props:atm/1)

set_env/2: PREDICATE

`set_env(Name,Value)`

The environment variable `Name` is assigned `Value`.

Usage:

– *The following properties should hold at call time:*

`Name` is an atom. (basic_props:atm/1)

`Value` is an atom. (basic_props:atm/1)

del_env/1: PREDICATE

`del_env(Name)`

The environment variable `Name` is removed.

Usage:

– *The following properties should hold at call time:*

`Name` is an atom. (basic_props:atm/1)

c_errno/1: PREDICATE

Usage:

– *Calls should, and exit will be compatible with:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Arg1` is an integer. (basic_props:int/1)

copy_file/2: PREDICATE

`copy_file(Source,Destination)`

Copies the file `Source` to `Destination`.

Usage:

– *The following properties should hold at call time:*

`Source` is an atom. (basic_props:atm/1)

`Destination` is an atom. (basic_props:atm/1)

copy_file/3: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an atom. (basic_props:atm/1)

`Arg2` is an atom. (basic_props:atm/1)

system:copy_options(Arg3) (system:copy_options/1)

dir_path/2: PREDICATE

No further documentation available for this predicate.

extract_paths/2: PREDICATE

`extract_paths(String,Paths)`

Interpret `String` as the value of a UNIX environment variable holding a list of paths and return in `Paths` the list of the paths. Paths in `String` are separated by colons, and an empty path is considered a shorthand for '.' (current path). The most typical environment variable with this format is PATH. For example, this is a typical use:

```
?- set_prolog_flag(write_strings, on).

yes
?- getenvstr('PATH', PATH), extract_paths(PATH, Paths).

PATH = ":/home/bardo/bin:/home/clip/bin:/opt/bin/:/bin",
Paths = [".","/home/bardo/bin","/home/clip/bin","/opt/bin/","/bin"] ?

yes
?-
```

Usage:

– *Call and exit should be compatible with:*

`Paths` is a list of `strings`. (basic_props:list/2)

– *The following properties should hold at call time:*

`String` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold upon exit:*

`Paths` is a list of `strings`. (basic_props:list/2)

file_dir_name/3: PREDICATE

`file_dir_name(File,Dir,Name)`

Discomposes a given File in its directory and name

Usage:

– *Calls should, and exit will be compatible with:*

`File` is an atom. (basic_props:atm/1)

`Dir` is an atom. (basic_props:atm/1)

`Name` is an atom. (basic_props:atm/1)

– *The following properties hold upon exit:*

`File` is an atom. (basic_props:atm/1)

`Dir` is an atom. (basic_props:atm/1)

`Name` is an atom. (basic_props:atm/1)

get_pid/1: PREDICATE

`get_pid(Pid)`

Unifies `Pid` with the process identificator of the current process or thread.

Usage:

– *Calls should, and exit will be compatible with:*

`Pid` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Pid` is an integer. (basic_props:int/1)

get_uid/1: PREDICATE

`get_uid(Uid)`

Unifies `Uid` with the user id of the current process.

Usage:

– *Calls should, and exit will be compatible with:*
`Uid` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*
`Uid` is an integer. (basic_props:int/1)

get_gid/1: PREDICATE

`get_gid(Uid)`

Unifies `Gid` with the group id of the current process.

Usage:

– *Calls should, and exit will be compatible with:*
`Uid` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*
`Uid` is an integer. (basic_props:int/1)

get_pwnam/1: PREDICATE

`get_pwnam(User)`

Unifies `User` with the user of the current process, as specified in the /etc/passwd file.

Usage:

– *Calls should, and exit will be compatible with:*
`User` is an atom. (basic_props:atm/1)

– *The following properties hold upon exit:*
`User` is an atom. (basic_props:atm/1)

get_grnam/1: PREDICATE

`get_grnam(Group)`

Unifies `Group` with the group of the current process, as specified in the /etc/group file.

Usage:

– *Calls should, and exit will be compatible with:*
`Group` is an atom. (basic_props:atm/1)

– *The following properties hold upon exit:*
`Group` is an atom. (basic_props:atm/1)

get_tmp_dir/1: PREDICATE

Usage: `get_tmp_dir(TmpDir)`

– *Description:* Gets the directory name used to store temporary files. In Unix is /tmp, in Windows is determined by the *TMP* environment variable.

– *The following properties should hold at call time:*
`TmpDir` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`TmpDir` is an atom. (basic_props:atm/1)

get_address/2: PREDICATE
No further documentation available for this predicate.

current_host/1: PREDICATE
`current_host(Hostname)`
`Hostname` is unified with the fully qualified name of the host.
Usage:
– *Calls should, and exit will be compatible with:*
`Hostname` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Hostname` is an atom. (basic_props:atm/1)

current_executable/1: PREDICATE
`current_executable(Path)`
Unifies `Path` with the path to the current executable.
Usage:
– *Calls should, and exit will be compatible with:*
`Path` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Path` is an atom. (basic_props:atm/1)

umask/2: PREDICATE
Usage 1: `umask(OldMask,NewMask)`
– *Description:* The process file creation mask was `OldMask`, and it is changed to `NewMask`.
– *The following properties should hold at call time:*
`NewMask` is an integer. (basic_props:int/1)
– *The following properties hold upon exit:*
`OldMask` is an integer. (basic_props:int/1)

Usage 2: `umask(OldMask,NewMask)`
– *Description:* Gets the process file creation mask without changing it.
– *The following properties should hold at call time:*
`OldMask` is a free variable. (term_typing:var/1)
`NewMask` is a free variable. (term_typing:var/1)
The terms `OldMask` and `NewMask` are strictly identical. (term_compare:== /2)
– *The following properties hold upon exit:*
`OldMask` is an integer. (basic_props:int/1)
`NewMask` is an integer. (basic_props:int/1)

make_directory/2: PREDICATE

`make_directory(DirName,Mode)`

Creates the directory `DirName` with a given `Mode`. This is, as usual, operated against the current umask value.

Usage:

– *The following properties should hold at call time:*

`DirName` is an atom. (basic_props:atm/1)

`Mode` is an integer. (basic_props:int/1)

make_directory/1: PREDICATE

`make_directory(DirName)`

Equivalent to `make_directory(D,0o777)`.

Usage:

– *The following properties should hold at call time:*

`DirName` is an atom. (basic_props:atm/1)

make_dirpath/2: PREDICATE

`make_dirpath(Path,Mode)`

Creates the whole `Path` for a given directory with a given `Mode`. As an example, `make_dirpath('/tmp/var/mydir/otherdir')`.

Usage:

– *The following properties should hold at call time:*

`Path` is a source name. (streams_basic:sourcename/1)

`Mode` is an integer. (basic_props:int/1)

make_dirpath/1: PREDICATE

`make_dirpath(Path)`

Equivalent to `make_dirpath(D,0o777)`.

Usage:

– *The following properties should hold at call time:*

`Path` is an atom. (basic_props:atm/1)

working_directory/2: PREDICATE

`working_directory(OldDir,NewDir)`

Unifies current working directory with `OldDir`, and then changes the working directory to `NewDir`. Calling `working_directory(Dir,Dir)` simply unifies `Dir` with the current working directory without changing anything else.

Usage 1:

– *Description:* Changes current working directory.

– *Calls should, and exit will be compatible with:*

`OldDir` is an atom. (basic_props:atm/1)

– *The following properties should hold at call time:*
`NewDir` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`OldDir` is an atom. (basic_props:atm/1)

Usage 2: `working_directory(OldDir,NewDir)`
– *Description:* Gets current working directory.
– *The following properties should hold at call time:*
`OldDir` is a free variable. (term_typing:var/1)
`NewDir` is a free variable. (term_typing:var/1)
The terms `OldDir` and `NewDir` are strictly identical. (term_compare:== /2)
– *The following properties hold upon exit:*
`OldDir` is an atom. (basic_props:atm/1)
`NewDir` is an atom. (basic_props:atm/1)

cd/1: PREDICATE

`cd(Path)`

Changes working directory to `Path`.

Usage:
– *The following properties should hold at call time:*
`Path` is an atom. (basic_props:atm/1)

shell/0: PREDICATE

Usage:
– *Description:* Execs the shell specified by the environment variable `SHELL`. When the shell process terminates, control is returned to Prolog.

shell/1: PREDICATE

`shell(Command)`

`Command` is executed in the shell specified by the environment variable `SHELL`. It succeeds if the exit code is zero and fails otherwise.

Usage:
– *The following properties should hold at call time:*
`Command` is an atom. (basic_props:atm/1)

shell/2: PREDICATE

`shell(Command,ReturnCode)`

Executes `Command` in the shell specified by the environment variable `SHELL` and stores the exit code in `ReturnCode`.

Usage:
– *Calls should, and exit will be compatible with:*
`ReturnCode` is an integer. (basic_props:int/1)

– *The following properties should hold at call time:*
`Command` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`ReturnCode` is an integer. (basic_props:int/1)

system/1: PREDICATE

`system(Command)`

Executes `Command` using the shell `/bin/sh`.

Usage:

– *The following properties should hold at call time:*
`Command` is an atom. (basic_props:atm/1)

system/2: PREDICATE

`system(Command,ReturnStatus)`

Executes `Command` in the `/bin/sh` shell and stores the return status in `ReturnStatus`. Note that the exit code is masked as the low order 8 bits of the return status:
`ReturnCode` is (`ReturnStatus` /\ 0xFF00) >> 8.

Usage:

– *Calls should, and exit will be compatible with:*
`ReturnStatus` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`Command` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`ReturnStatus` is an integer. (basic_props:int/1)

popen/3: PREDICATE

`popen(Command,Mode,Stream)`

Open a pipe to process `Command` in a new shell with a given `Mode` and return a communication `Stream` (as in UNIX `popen(3)`). If `Mode` is `read` the output from the process is sent to `Stream`. If `Mode` is `write`, `Stream` is sent as input to the process. `Stream` may be read from or written into using the ordinary stream I/O predicates. `Stream` must be closed explicitly using `close/1`, i.e., it is not closed automatically when the process dies. Note that `popen/2` is defined in ***x as using `/bin/sh`, which usually does not exist in Windows systems. In this case, a `sh` shell which comes with Windows is used.

Usage:

– *The following properties should hold at call time:*
`Command` is an atom. (basic_props:atm/1)
`Mode` is 'read' or 'write'. (system:popen_mode/1)
`Stream` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Stream` is an open stream. (streams_basic:stream/1)

popen_mode/1: REGTYPE

Usage: `popen_mode(M)`

– *Description:* `M` is 'read' or 'write'.

exec/4: PREDICATE

`exec(Command,StdIn,StdOut,StdErr)`

Starts the process `Command` and returns the standart I/O streams of the process in `StdIn`, `StdOut`, and `StdErr`. If `Command` contains blank spaces, these are taken as separators between a program name (the first chunk of contiguous non-blank characters) and options for the program (the subsequent contiguous pieces of non-blank characters), as in `exec('ls -lRa ../sibling_dir', In, Out, Err)`.

Usage:

– *The following properties should hold at call time:*

`Command` is an atom. (basic_props:atm/1)

`StdIn` is a free variable. (term_typing:var/1)

`StdOut` is a free variable. (term_typing:var/1)

`StdErr` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`StdIn` is an open stream. (streams_basic:stream/1)

`StdOut` is an open stream. (streams_basic:stream/1)

`StdErr` is an open stream. (streams_basic:stream/1)

exec/3: PREDICATE

`exec(Command,StdIn,StdOut)`

Starts the process `Command` and returns the standart I/O streams of the process in `StdIn` and `StdOut`. `Standard error` is connected to whichever the parent process had it connected to. `Command` is treated and split in components as in `exec/4`.

Usage:

– *The following properties should hold at call time:*

`Command` is an atom. (basic_props:atm/1)

`StdIn` is a free variable. (term_typing:var/1)

`StdOut` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`StdIn` is an open stream. (streams_basic:stream/1)

`StdOut` is an open stream. (streams_basic:stream/1)

exec/8: PREDICATE

Usage: `exec(Command,Arguments,StdIn,StdOut,StdErr,Background,PID,ErrCode)`

– *Description:* `exec/8` gives a finer control on execution of process. `Command` is the command to be executed and `Arguments` is a list of atoms to be passed as arguments to the command. When called with free variables, `StdIn`, `StdOut`, and `StdErr` are instantiated to streams connected to the standard output, input, and error of the created process. `Background` controls whether the caller waits for `Command` to finish,

or if the process executing `Command` is completely detached (it can be waited for using `wait/3`). `ErrCode` is the error code returned by the lower-level `exec()` system call (this return code is system-dependent, but a non-zero value usually means that something has gone wrong). If `Command` does not start by a slash, `exec/8` uses the environment variable `PATH` to search for it. If `PATH` is not set, `/bin` and `/usr/bin` are searched.

– *The following properties should hold at call time:*

`Command` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Arguments` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Background` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`PID` is a free variable.	(term_typing:var/1)
`ErrCode` is a free variable.	(term_typing:var/1)
`Command` is an atom.	(basic_props:atm/1)
`Arguments` is a list of `atm`s.	(basic_props:list/2)
`Background` is an atom.	(basic_props:atm/1)

– *The following properties should hold upon exit:*

`StdIn` is an open stream.	(streams_basic:stream/1)
`StdOut` is an open stream.	(streams_basic:stream/1)
`StdErr` is an open stream.	(streams_basic:stream/1)
`PID` is an integer.	(basic_props:int/1)
`ErrCode` is an integer.	(basic_props:int/1)

wait/3: PREDICATE

Usage: `wait(Pid,RetCode,Status)`

– *Description:* wait/3 waits for the process numbered `Pid`. If `PID` equals -1, it will wait for any children process. `RetCode` is usually the PID of the waited-for process, and -1 in case in case of error. `Status` is related to the exit value of the process in a system-dependent fashion.

– *The following properties should hold at call time:*

`Pid` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`RetCode` is a free variable.	(term_typing:var/1)
`Status` is a free variable.	(term_typing:var/1)
`Pid` is an integer.	(basic_props:int/1)
`RetCode` is a free variable.	(term_typing:var/1)
`Status` is a free variable.	(term_typing:var/1)

– *The following properties hold upon exit:*

`RetCode` is an integer.	(basic_props:int/1)
`Status` is an integer.	(basic_props:int/1)

directory_files/2: PREDICATE

`directory_files(Directory,FileList)`

`FileList` is the unordered list of entries (files, directories, etc.) in `Directory`.

Usage:

– *Calls should, and exit will be compatible with:*
`FileList` is a list of `atm`s. (basic_props:list/2)
– *The following properties should hold at call time:*
`Directory` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`FileList` is a list of `atm`s. (basic_props:list/2)

mktemp/2: PREDICATE

`mktemp(Template,Filename)`

Returns a unique `Filename` based on `Template`: `Template` must be a valid file name with six trailing X, which are substituted to create a new file name. `Filename` is created in read/write mode but closed immediately after creation.

Usage:

– *Calls should, and exit will be compatible with:*
`Filename` is an atom. (basic_props:atm/1)
– *The following properties should hold at call time:*
`Template` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Filename` is an atom. (basic_props:atm/1)

mktemp_in_tmp/2: PREDICATE

No further documentation available for this predicate.

file_exists/1: PREDICATE

`file_exists(File)`

Succeeds if `File` (a file or directory) exists (and is accessible).

Usage:

– *The following properties should hold at call time:*
`File` is an atom. (basic_props:atm/1)

file_exists/2: PREDICATE

`file_exists(File,Mode)`

`File` (a file or directory) exists and it is accessible with `Mode`, as in the Unix call `access(2)`. Typically, `Mode` is 4 for read permission, 2 for write permission and 1 for execute permission.

Usage:

– *The following properties should hold at call time:*
`File` is an atom. (basic_props:atm/1)
`Mode` is an integer. (basic_props:int/1)
– *The following properties hold upon exit:*
`File` is an atom. (basic_props:atm/1)
`Mode` is an integer. (basic_props:int/1)

file_property/2: PREDICATE

`file_property(File,Property)`

`File` has the property `Property`. The possible properties are:

type(`Type`)
: `Type` is one of `regular`, `directory`, `symlink`, `fifo`, `socket` or `unknown`.

linkto(`Linkto`)
: If `File` is a symbolic link, `Linkto` is the file pointed to by the link (and the other properties come from that file, not from the link itself).

mod_time(`ModTime`)
: `ModTime` is the time of last modification (seconds since January, 1, 1970).

mode(`Protection`)
: `Protection` is the protection mode.

size(`Size`) `Size` is the size.

If `Property` is uninstantiated, the predicate will enumerate the properties on backtracking.

Usage:

– *Call and exit should be compatible with:*

`Property` is a compound term. (basic_props:struct/1)

– *The following properties should hold at call time:*

`File` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*

`Property` is a compound term. (basic_props:struct/1)

file_properties/6: PREDICATE

`file_properties(Path,Type,Linkto,Time,Protection,Size)`

The file `Path` has the following properties:

- File type `Type` (one of `regular`, `directory`, `symlink`, `fifo`, `socket` or `unknown`).
- If `Path` is a symbolic link, `Linkto` is the file pointed to. All other properties come from the file pointed, not the link. `Linkto` is '' if `Path` is not a symbolic link.
- Time of last modification `Time` (seconds since January, 1, 1970).
- Protection mode `Protection`.
- Size in bytes `Size`.

Usage:

– *Calls should, and exit will be compatible with:*

`Type` is an atom. (basic_props:atm/1)

`Linkto` is an atom. (basic_props:atm/1)

`Time` is an integer. (basic_props:int/1)

`Protection` is an integer. (basic_props:int/1)

`Size` is an integer. (basic_props:int/1)

– *The following properties should hold at call time:*

`Path` is an atom. (basic_props:atm/1)

– *The following properties hold upon exit:*
`Type` is an atom. (basic_props:atm/1)
`Linkto` is an atom. (basic_props:atm/1)
`Time` is an integer. (basic_props:int/1)
`Protection` is an integer. (basic_props:int/1)
`Size` is an integer. (basic_props:int/1)

modif_time/2: PREDICATE

`modif_time(File,Time)`

The file `File` was last modified at `Time`, which is in seconds since January, 1, 1970. Fails if `File` does not exist.

Usage:

– *Call and exit should be compatible with:*
`Time` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`File` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
`Time` is an integer. (basic_props:int/1)

modif_time0/2: PREDICATE

`modif_time0(File,Time)`

If `File` exists, `Time` is its latest modification time, as in `modif_time/2`. Otherwise, if `File` does not exist, `Time` is zero.

Usage:

– *Call and exit should be compatible with:*
`Time` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`File` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
`Time` is an integer. (basic_props:int/1)

fmode/2: PREDICATE

`fmode(File,Mode)`

The file `File` has protection mode `Mode`.

Usage:

– *Call and exit should be compatible with:*
`Mode` is an integer. (basic_props:int/1)
– *The following properties should hold at call time:*
`File` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
`Mode` is an integer. (basic_props:int/1)

chmod/2: PREDICATE

`chmod(File,NewMode)`

Change the protection mode of file `File` to `NewMode`.

Usage:

– *The following properties should hold at call time:*

`File` is an atom. (basic_props:atm/1)

`NewMode` is an integer. (basic_props:int/1)

chmod/3: PREDICATE

`chmod(File,OldMode,NewMode)`

The file `File` has protection mode `OldMode` and it is changed to `NewMode`.

Usage 1:

– *Call and exit should be compatible with:*

`OldMode` is an integer. (basic_props:int/1)

– *The following properties should hold at call time:*

`File` is an atom. (basic_props:atm/1)

`NewMode` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`OldMode` is an integer. (basic_props:int/1)

Usage 2: `chmod(File,OldMode,NewMode)`

– *Description:* If `OldMode` is identical to `NewMode` then it is equivalent to fmode(`File`,`OldMode`)

– *The following properties should hold at call time:*

`File` is an atom. (basic_props:atm/1)

`OldMode` is a free variable. (term_typing:var/1)

`NewMode` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`File` is an atom. (basic_props:atm/1)

`OldMode` is an atom. (basic_props:atm/1)

`NewMode` is an atom. (basic_props:atm/1)

set_exec_mode/2: PREDICATE

`set_exec_mode(SourceName,ExecName)`

Copies the permissions of `SourceName` to `ExecName` adding permissions to execute.

Usage:

– *The following properties should hold at call time:*

`SourceName` is an atom. (basic_props:atm/1)

`ExecName` is an atom. (basic_props:atm/1)

delete_file/1: PREDICATE

`delete_file(File)`

Delete the file `File`.

Usage:

– *The following properties should hold at call time:*

`File` is an atom. (basic_props:atm/1)

delete_directory/1: PREDICATE

`delete_directory(File)`

Delete the directory `Directory`.

Usage:

– *The following properties should hold at call time:*

`File` is an atom. (basic_props:atm/1)

rename_file/2: PREDICATE

`rename_file(File1,File2)`

Change the name of `File1` to `File2`.

Usage:

– *The following properties should hold at call time:*

`File1` is an atom. (basic_props:atm/1)

`File2` is an atom. (basic_props:atm/1)

using_windows/0: PREDICATE

Usage:

– *Description:* Success if the operating system is using windows instead of a posix operating system (which includes cygwin under windows). To do this, we look at the CIAOSCRIPT environment variable, so if it is defined we suppose we are inside a posix operating system (and not windows).

winpath/2: PREDICATE

Usage 1: `winpath(A,B)`

– *The following properties should hold at call time:*

`A` is an atom. (basic_props:atm/1)

`B` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`A` is an atom. (basic_props:atm/1)

`B` is an atom. (basic_props:atm/1)

Usage 2: `winpath(A,B)`

– *The following properties should hold at call time:*

`A` is a free variable. (term_typing:var/1)

`B` is an atom. (basic_props:atm/1)

– *The following properties should hold upon exit:*
`A` is an atom. (basic_props:atm/1)
`B` is an atom. (basic_props:atm/1)

Usage 3: `winpath(A,B)`
– *The following properties should hold at call time:*
`A` is an atom. (basic_props:atm/1)
`B` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
`A` is an atom. (basic_props:atm/1)
`B` is an atom. (basic_props:atm/1)

winpath/3: PREDICATE

`winpath(Option,Posix,WinPath)`

`Option` specifies if you want to get a relative or a full path. `Posix` represent a path as usual in unix, and `WinPath` is the Windows-Style representation of `Posix`.

Usage 1:
– *The following properties should hold at call time:*
`Option` is a free variable. (term_typing:var/1)
`Posix` is a free variable. (term_typing:var/1)
`WinPath` is an atom. (basic_props:atm/1)
– *The following properties should hold upon exit:*
system:winpath_option(Option) (system:winpath_option/1)
`Posix` is an atom. (basic_props:atm/1)

Usage 2:
– *The following properties should hold at call time:*
`Option` is a free variable. (term_typing:var/1)
`Posix` is an atom. (basic_props:atm/1)
`WinPath` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
system:winpath_option(Option) (system:winpath_option/1)
`WinPath` is an atom. (basic_props:atm/1)

winpath_c/3: PREDICATE

Same as winpath/3, but for strings.

cyg2win/3: PREDICATE

Usage: `cyg2win(CygWinPath,WindowsPath,SwapSlash)`
– *Description:* Converts a posix path to a Windows-style path. If `SwapSlash` is `swap`, slashes are converted in to backslash. If it is `noswap`, they are preserved.

– *The following properties should hold at call time:*
`CygWinPath` is a string (a list of character codes). (basic_props:string/1)
`WindowsPath` is a free variable. (term_typing:var/1)
`SwapSlash` is currently instantiated to an atom. (term_typing:atom/1)
– *The following properties should hold upon exit:*
`CygWinPath` is a string (a list of character codes). (basic_props:string/1)
`WindowsPath` is a string (a list of character codes). (basic_props:string/1)
`SwapSlash` is currently instantiated to an atom. (term_typing:atom/1)

no_swapslash/3: PREDICATE

No further documentation available for this predicate.

replace_characters/4: PREDICATE

`replace_characters(String,SearchChar,ReplaceChar,Output)`

Replaces all the occurrences of `SearchChar` by `ReplaceChar` and unifies the result with `Output`

system_error_report/1: PREDICATE

Usage: `system_error_report(Report)`

– *Description:* Report is the error message from the last system call, like `strerror` in POSIX.
– *The following properties should hold at call time:*
`Report` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Report` is an atom. (basic_props:atm/1)

48.3 Documentation on multifiles (`system`)

define_flag/3: PREDICATE

The predicate is *multifile*.

Usage: `define_flag(Flag,FlagValues,Default)`

– *The following properties hold upon exit:*
`Flag` is an atom. (basic_props:atm/1)
Define the valid flag values (basic_props:flag_values/1)

48.4 Known bugs and planned improvements (`system`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.
- In some situations, copy_file don't work when the second argument is a directory, example: copy_file('site/ciaopp_online.html' , ~distdir, yes),
- `shell/n` commands have a bug in Windows: if the environment variable SHELL is instantiated to some Windows shell implementation, then it is very possible that shell/`{1,2}` will not work, as it is always called with the -c flag to start the user command. For example, COMMAND.COM **might** need the flag /C – but there is no way to know a priori which command line option is necessary for every shell! It does not seems usual that Windows sets the SHELL environment variable: if it is not set, we set it up at startup time to point to the `sh.exe` provided with Ciao, which is able to start Windows aplications. Therefore, `?- shell('command.com').` just works.
- If `exec/4` does not find the command to be executed, there is no visible error message: it is sent to a error output which has already been assigned to a different stream, disconnected from the one the user sees.

49 Prolog system internal predicates

Author(s): Manuel Carro, Daniel Cabeza, José F. Morales, Mats Carlsson (original author).

This module implements some miscellaneous predicates which provide access to some internal statistics, special properties of the predicates, etc.

49.1 Usage and interface (prolog_sys)

- **Library usage:**
  `:- use_module(library(prolog_sys)).`
- **Exports:**
  - *Predicates:*
    `statistics/0`, `statistics/2`, `new_atom/1`, `predicate_property/2`, `predicate_property/3`, `current_atom/1`, `garbage_collect/0`.
  - *Regular Types:*
    `clockfreq_result/1`, `tick_result/1`, `symbol_result/1`, `gc_result/1`, `memory_result/1`, `time_result/1`, `symbol_option/1`, `garbage_collection_option/1`, `memory_option/1`, `clockfreq_option/1`, `tick_option/1`, `time_option/1`.

49.2 Documentation on exports (prolog_sys)

statistics/0: PREDICATE

Usage:

- *Description:* Prints statistics about the system.

statistics/2: PREDICATE

Usage 1: `statistics(Tick_option,Tick_result)`

- *Description:* Gather information about clock ticks (either run, user, system or wall tick) since last consult or since start of program. A tick is the smallest amount of time that a clock can measure.
- *The following properties should hold at call time:*
  Options to get information about execution ticks. (prolog_sys:tick_option/1)
  `Tick_result` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*
  Options to get information about execution ticks. (prolog_sys:tick_option/1)
  `Tick_result` is a two-element list of numbers. The first number is the number of ticks since the start of the execution; the second number is the number of ticks since the previous consult to tick. (prolog_sys:tick_result/1)

Usage 2: `statistics(Clockfreq_option,Clockfreq_result)`

- *Description:* Gather information about frequency of the clocks used to measure the ticks (either run-user, system or wall clock). Results are returned in hertz. This value also can be defined as the amount of ticks that a clock can measure in one second.

– *The following properties should hold at call time:*

Options to get information about the frequency of clocks used to get the ticks. (prolog_sys:clockfreq_option/1)

`Clockfreq_result` is any term. (basic_props:term/1)

– *The following properties should hold upon exit:*

Options to get information about the frequency of clocks used to get the ticks. (prolog_sys:clockfreq_option/1)

`Clockfreq_result` is a number. It gives the frequency in hertz used by the clock get the ticks. (prolog_sys:clockfreq_result/1)

Usage 3: `statistics(Time_option,Time_result)`

– *Description:* Gather information about time (either process time or wall time) since last consult or since start of program. Results are returned in milliseconds. Note that internally, time is calculated as:

```
Time_result = (Tick_result / Clockfreq_result) * 1000
```

– *The following properties should hold at call time:*

Options to get information about execution time. `Time_option` must be one of `runtime`, `usertime`, `systemtime` or `walltime`. (prolog_sys:time_option/1)

`Time_result` is any term. (basic_props:term/1)

– *The following properties should hold upon exit:*

Options to get information about execution time. `Time_option` must be one of `runtime`, `usertime`, `systemtime` or `walltime`. (prolog_sys:time_option/1)

`Time_result` is a two-element list of numbers. The first number is the time since the start of the execution; the second number is the time since the previous consult to time. (prolog_sys:time_result/1)

Usage 4: `statistics(Memory_option,Memory_result)`

– *Description:* Gather information about memory consumption.

– *The following properties should hold at call time:*

Options to get information about memory usage. (prolog_sys:memory_option/1)

`Memory_result` is any term. (basic_props:term/1)

– *The following properties should hold upon exit:*

Options to get information about memory usage. (prolog_sys:memory_option/1)

Result is a two-element list of integers. The first element is the space taken up by the option selected, measured in bytes; the second integer is zero for program space (which grows as necessary), and the amount of free space otherwise. (prolog_sys:memory_result/1)

Usage 5: `statistics(Garbage_collection_option,Gc_result)`

– *Description:* Gather information about garbage collection.

– *The following properties should hold at call time:*

Options to get information about garbage collection. (prolog_sys:garbage_collection_option/1)

`Gc_result` is any term. (basic_props:term/1)

– *The following properties should hold upon exit:*

Options to get information about garbage collection. (prolog_sys:garbage_collection_option/1)

`Gc_result` is a tree-element list of integers, related to garbage collection and memory management. When `stack_shifts` is selected, the first one is the number of shifts

(reallocations) of the local stack; the second is the number of shifts of the trail, and the third is the time spent in these shifts. When `garbage_collection` is selected, the numbers are, respectively, the number of garbage collections performed, the number of bytes freed, and the time spent in garbage collection. (prolog_sys:gc_result/1)

Usage 6: `statistics(Symbol_option,Symbol_result)`

- *Description:* Gather information about number of symbols and predicates.
- *The following properties should hold at call time:*

  Option to get information about the number of symbols in the program. (prolog_sys:symbol_option/1)

  `Symbol_result` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*

  Option to get information about the number of symbols in the program. (prolog_sys:symbol_option/1)

  `Symbol_result` is a two-element list of integers. The first one is the number of atom, functor, and predicate names in the symbol table. The second is the number of predicates known to be defined (although maybe without clauses). (prolog_sys:symbol_result/1)

Usage 7: `statistics(Option,Arg2)`

- *Description:* If `Option` is unbound, it is bound to the values on the other cases.
- *Call and exit should be compatible with:*

  `Arg2` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*

  `Arg2` is any term. (basic_props:term/1)

clockfreq_result/1: REGTYPE

Usage: `clockfreq_result(Result)`

- *Description:* `Result` is a number. It gives the frequency in hertz used by the clock get the ticks.

tick_result/1: REGTYPE

Usage: `tick_result(Result)`

- *Description:* `Result` is a two-element list of numbers. The first number is the number of ticks since the start of the execution; the second number is the number of ticks since the previous consult to tick.

new_atom/1: PREDICATE

Usage: `new_atom(Atom)`

- *Description:* Returns, on success, a new atom, not existing before in the system. The entry argument must be a variable. The idea behind this atom generation is to provide a fast source of identifiers for new objects, concurrent predicates, etc. on the fly.
- *The following properties should hold at call time:*

  `Atom` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*

  `Atom` is an atom. (basic_props:atm/1)

symbol_result/1: REGTYPE

Usage: `symbol_result(Result)`

– *Description:* `Result` is a two-element list of integers. The first one is the number of atom, functor, and predicate names in the symbol table. The second is the number of predicates known to be defined (although maybe without clauses).

gc_result/1: REGTYPE

Usage: `gc_result(Result)`

– *Description:* `Result` is a tree-element list of integers, related to garbage collection and memory management. When `stack_shifts` is selected, the first one is the number of shifts (reallocations) of the local stack; the second is the number of shifts of the trail, and the third is the time spent in these shifts. When `garbage_collection` is selected, the numbers are, respectively, the number of garbage collections performed, the number of bytes freed, and the time spent in garbage collection.

memory_result/1: REGTYPE

Usage: `memory_result(Result)`

– *Description:* Result is a two-element list of integers. The first element is the space taken up by the option selected, measured in bytes; the second integer is zero for program space (which grows as necessary), and the amount of free space otherwise.

time_result/1: REGTYPE

Usage: `time_result(Result)`

– *Description:* `Result` is a two-element list of numbers. The first number is the time since the start of the execution; the second number is the time since the previous consult to time.

symbol_option/1: REGTYPE

Usage: `symbol_option(M)`

– *Description:* Option to get information about the number of symbols in the program.

garbage_collection_option/1: REGTYPE

Usage: `garbage_collection_option(M)`

– *Description:* Options to get information about garbage collection.

memory_option/1: REGTYPE

Usage: `memory_option(M)`

– *Description:* Options to get information about memory usage.

clockfreq_option/1: REGTYPE

Usage: `clockfreq_option(M)`

– *Description:* Options to get information about the frequency of clocks used to get the ticks.

tick_option/1: REGTYPE

Usage: `tick_option(M)`

– *Description:* Options to get information about execution ticks.

time_option/1: REGTYPE

Usage: `time_option(M)`

– *Description:* Options to get information about execution time. `M` must be one of `runtime`, `usertime`, `systemtime` or `walltime`.

predicate_property/2: PREDICATE

General properties: `predicate_property(Head,Prop)`

– *Description:* Predicate property of `true/0` is compiled

– *If the following properties should hold at call time:*

term_basic:Head=true (term_basic:= /2)

then the following properties should hold upon exit:

term_basic:Prop=compiled (term_basic:= /2)

Usage: `predicate_property(Head,Property)`

– *Description:* The predicate `Head`, visible from the current module, (a goal) has the property `Property`.

– *The following properties should hold upon exit:*

`Head` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Property` is an atom. (basic_props:atm/1)

predicate_property/3: PREDICATE

No further documentation available for this predicate.

current_atom/1: PREDICATE

Usage: `current_atom(Atom)`

– *Description:* Enumerates on backtracking all the existing atoms in the system.

– *The following properties hold upon exit:*

`Atom` is an atom. (basic_props:atm/1)

garbage_collect/0: PREDICATE

Usage:

– *Description:* Forces garbage collection when called.

49.3 Known bugs and planned improvements (prolog_sys)

- The space used by the process is not measured here: process data, code, and stack also take up memory. The memory reported for atoms is not what is actually used, but the space used up by the hash table (which is enlarged as needed).
- The predicate `predicate_property/2` needs more work:
  - Efficiency: In order to be complete and efficient, this needs to be a built-in predicate of our module system. Consulting predicate properties does not seem a dangerous operation (except that, when it cannot be resolved at compile-time, it prevents removal of module runtime information).
  - Correctness: The head is automatically module-expanded on call. If the head is not module-expanded, there are consistency problems. Other systems avoid those problems by disallowing the import of two predicates with the same name from different modules. That is clearly not a solution in Ciao.
- Implement a `'$predicate_property'/2` where the module can be specified. That will simplify the `predicate_property/2` implementation

50 DEC-10 Prolog file IO

This module implements the support for DEC-10 Prolog style file I/O.

50.1 Usage and interface (dec10_io)

- **Library usage:**
  `:- use_module(library(dec10_io)).`
- **Exports:**
  - *Predicates:*
    `see/1`, `seeing/1`, `seen/0`, `tell/1`, `telling/1`, `told/0`, `close_file/1`.

50.2 Documentation on exports (dec10_io)

see/1: PREDICATE

Usage: `see(File)`

- *The following properties should hold at call time:*
  `File` is currently instantiated to an atom. (term_typing:atom/1)

seeing/1: PREDICATE

Usage: `seeing(File)`

- *The following properties should hold upon exit:*
  `File` is currently instantiated to an atom. (term_typing:atom/1)

seen/0: PREDICATE

No further documentation available for this predicate.

tell/1: PREDICATE

Usage: `tell(File)`

- *The following properties should hold at call time:*
  `File` is currently instantiated to an atom. (term_typing:atom/1)

telling/1: PREDICATE

Usage: `telling(File)`

- *The following properties should hold upon exit:*
  `File` is currently instantiated to an atom. (term_typing:atom/1)

told/0: PREDICATE

No further documentation available for this predicate.

close_file/1: PREDICATE

No further documentation available for this predicate.

51 Quintus-like internal database

Author(s): The CLIP Group.

The predicates described in this section were introduced in early implementations of Prolog to provide efficient means of performing operations on large quantities of data. The introduction of indexed dynamic predicates have rendered these predicates obsolete, and the sole purpose of providing them is to support existing code. There is no reason whatsoever to use them in new code.

These predicates store arbitrary terms in the database without interfering with the clauses which make up the program. The terms which are stored in this way can subsequently be retrieved via the key on which they were stored. Many terms may be stored on the same key, and they can be individually accessed by pattern matching. Alternatively, access can be achieved via a special identifier which uniquely identifies each recorded term and which is returned when the term is stored.

51.1 Usage and interface (`old_database`)

- **Library usage:**
  `:- use_module(library(old_database)).`
- **Exports:**
  - *Predicates:*
    `recorda/3`, `recordz/3`, `recorded/3`, `current_key/2`.

51.2 Documentation on exports (`old_database`)

recorda/3: PREDICATE

`recorda(Key,Term,Ref)`

The term `Term` is recorded in the internal database as the first item for the key `Key`, where `Ref` is its implementation-defined identifier. The key must be given, and only its principal functor is significant. Any uninstantiated variables in the `Term` will be replaced by new private variables, along with copies of any subgoals blocked on these variables.

Usage: `recorda(Key,Term,Ref)`

- *The following properties should hold at call time:*
  `Key` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Ref` is a free variable. (term_typing:var/1)
- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

recordz/3: PREDICATE

`recordz(Key,Term,Ref)`

Like `recorda/3`, except that the new term becomes the *last* item for the key `Key`.

Usage: `recordz(Key,Term,Ref)`

– *The following properties should hold at call time:*
`Key` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Ref` is a free variable. (term_typing:var/1)
– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

recorded/3: PREDICATE

`recorded(Key,Term,Ref)`

The internal database is searched for terms recorded under the key `Key`. These terms are successively unified with `Term` in the order they occur in the database. At the same time, `Ref` is unified with the implementation-defined identifier uniquely identifying the recorded item. If the key is instantiated to a compound term, only its principal functor is significant. If the key is uninstantiated, all terms in the database are successively unified with `Term` in the order they occur.

Usage: `recorded(Key,Term,Ref)`

– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

current_key/2: PREDICATE

`current_key(KeyName,KeyTerm)`

`KeyTerm` is the most general form of the key for a currently recorded term, and `KeyName` is the name of that key. This predicate can be used to enumerate in undefined order all keys for currently recorded terms through backtracking.

Usage: `current_key(Name,Key)`

– *The following properties should hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

52 ttyout (library)

52.1 Usage and interface (ttyout)

- **Library usage:**
  `:- use_module(library(ttyout)).`
- **Exports:**
  - *Predicates:*
    `ttyget/1`, `ttyget1/1`, `ttynl/0`, `ttyput/1`, `ttyskip/1`, `ttytab/1`, `ttyflush/0`, `ttydisplay/1`, `ttydisplayq/1`, `ttyskipeol/0`, `ttydisplay_string/1`.

52.2 Documentation on exports (ttyout)

ttyget/1: PREDICATE

Usage: `ttyget(X)`

- *The following properties should hold upon exit:*
  `X` is an integer. (basic_props:int/1)
- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

ttyget1/1: PREDICATE

No further documentation available for this predicate.

ttynl/0: PREDICATE

- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

ttyput/1: PREDICATE

Usage: `ttyput(X)`

- *The following properties should hold at call time:*
  `X` is an integer. (basic_props:int/1)
- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

ttyskip/1: PREDICATE

No further documentation available for this predicate.

ttytab/1: PREDICATE
No further documentation available for this predicate.

ttyflush/0: PREDICATE
– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

ttydisplay/1: PREDICATE
No further documentation available for this predicate.

ttydisplayq/1: PREDICATE
No further documentation available for this predicate.

ttyskipeol/0: PREDICATE
No further documentation available for this predicate.

ttydisplay_string/1: PREDICATE
No further documentation available for this predicate.

53 Enabling operators at run-time

Author(s): Daniel Cabeza.

This library package allows the use of the statically defined operators of a module for the reading performed at run-time by the program that uses the module. Simply by using this package the operator definitions appearing in the module are enabled during the execution of the program.

53.1 Usage and interface (`runtime_ops_doc`)

- **Library usage:**
  `:- use_package(runtime_ops).`
  or
  `:- module(...,...,[runtime_ops]).`
- **Other modules used:**
  - *System library modules:*
    `operators`.

PART V - Assertions, Properties, Types, Modes, Comments (assertions)

Author(s): The CLIP Group.

Ciao allows *annotating* the program code with *assertions*. Such assertions include type and instantiation mode declarations, but also more general properties as well as comments for *autodocumentation* in the *literate programming* style. These assertions document predicates (and modules and whole applications) and can be used by the Ciao preprocessor/compiler while debugging and optimizing the program or library, and by the Ciao documenter to build program or library reference manuals.

54 The Ciao assertion package

Author(s): Manuel Hermenegildo, Francisco Bueno, German Puebla.

The `assertions` package adds a number of new declaration definitions and new operator definitions which allow including program assertions in user programs. Such assertions can be used to describe predicates, properties, modules, applications, etc. These descriptions can contain formal specifications (such as sets of preconditions, post-conditions, or descriptions of computations) as well as machine-readable textual comments.

This module is part of the `assertions` library. It defines the basic code-related assertions, i.e., those intended to be used mainly by compilation-related tools, such as the static analyzer or the run-time test generator.

Giving specifications for predicates and other program elements is the main functionality documented here. The exact syntax of comments is described in the autodocumenter (`lpdoc` [Knu84,Her99]) manual, although some support for adding machine-readable comments in assertions is also mentioned here.

There are two kinds of assertions: predicate assertions and program point assertions. All predicate assertions are currently placed as directives in the source code, i.e., preceded by "`:-`". Program point assertions are placed as goals in clause bodies.

54.1 More info

The facilities provided by the library are documented in the description of its component modules. This documentation is intended to provide information only at a "reference manual" level. For a more tutorial introduction to the subject and some more examples please see [PBH00]. The assertion language implemented in this library is modeled after this design document, although, due to implementation issues, it may differ in some details. The purpose of this manual is to document precisely what the implementation of the library supports at any given point in time.

54.2 Some attention points

- **Formatting commands within text strings:** many of the predicates defined in these modules include arguments intended for providing textual information. This includes titles, descriptions, comments, etc. The type of this argument is a character string. In order for the automatic generation of documentation to work correctly, this character string should adhere to certain conventions. See the description of the `docstring/1` type/grammar for details.
- **Referring to variables:** In order for the automatic documentation system to work correctly, variable names (for example, when referring to arguments in the head patterns of *pred* declarations) must be surrounded by an `@var` command. For example, `@var{VariableName}` should be used for referring to the variable "VariableName", which will appear then formatted as follows: `VariableName`. See the description of the `docstring/1` type/grammar for details.

54.3 Usage and interface (assertions_doc)

- **Library usage:**
  The recommended procedure in order to make use of assertions in user programs is to include the `assertions` syntax library, using one of the following declarations, as appropriate:

```
:- module(...,...,[assertions]).
:- use_package([assertions]).
```

- **Exports:**
  - *Predicates:*
    `check/1`, `trust/1`, `true/1`, `false/1`.
- **New operators defined:**
  `=>/2` [975,xfx], `::/2` [978,xfx], `decl/1` [1150,fx], `decl/2` [1150,xfx], `pred/1` [1150,fx], `pred/2` [1150,xfx], `prop/1` [1150,fx], `prop/2` [1150,xfx], `modedef/1` [1150,fx], `calls/1` [1150,fx], `calls/2` [1150,xfx], `success/1` [1150,fx], `success/2` [1150,xfx], `test/1` [1150,fx], `test/2` [1150,xfx], `texec/1` [1150,fx], `texec/2` [1150,xfx], `comp/1` [1150,fx], `comp/2` [1150,xfx], `entry/1` [1150,fx], `exit/1` [1150,fx], `exit/2` [1150,xfx].
- **New declarations defined:**
  `pred/1`, `pred/2`, `texec/1`, `texec/2`, `calls/1`, `calls/2`, `success/1`, `success/2`, `test/1`, `test/2`, `comp/1`, `comp/2`, `prop/1`, `prop/2`, `entry/1`, `exit/1`, `exit/2`, `modedef/1`, `decl/1`, `decl/2`, `doc/2`, `comment/2`.
- **Other modules used:**
  - *System library modules:*
    `assertions/assertions_props`.

54.4 Documentation on new declarations (assertions_doc)

pred/1: DECLARATION

This assertion provides information on a predicate. The body of the assertion (its only argument) contains properties or comments in the formats defined by `assrt_body/1`.

More than one of these assertions may appear per predicate, in which case each one represents a possible " mode" of use (usage) of the predicate. The exact scope of the usage is defined by the properties given for calls in the body of each assertion (which should thus distinguish the different usages intended). All of them together cover all possible modes of usage.

For example, the following assertions describe (all the and the only) modes of usage of predicate `length/2` (see `lists`):

```
:- pred length(L,N) : list * var => list * integer
# "Computes the length of L.".
:- pred length(L,N) : var * integer => list * integer
# "Outputs L of length N.".
:- pred length(L,N) : list * integer => list * integer
# "Checks that L is of length N.".
```

Usage: :- `pred AssertionBody`.

- *The following properties should hold at call time:*
  `AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

pred/2: DECLARATION

This assertion is similar to a `pred/1` assertion but it is explicitely qualified. Non-qualified `pred/1` assertions are assumed the qualifier `check`.

Usage: :- `AssertionStatus pred AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

texec/1: DECLARATION

This assertion is similar to a `calls/1` assertion but it is used to provide input data and execution commands to the unit-test driver.

Usage: :- `texec AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is a call assertion body. (assertions_props:c_assrt_body/1)

texec/2: DECLARATION

This assertion is similar to a `texec/1` assertion but it is explicitely qualified with an assertion status. Non-qualified `texec/1` assertions are assumed to have `check` status.

Usage: :- `AssertionStatus texec AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is a call assertion body. (assertions_props:c_assrt_body/1)

calls/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it only provides information about the calls to a predicate. If one or several calls assertions are given they are understood to describe all possible calls to the predicate.

For example, the following assertion describes all possible calls to predicate `is/2` (see `arithmetic`):

```
:- calls is(term,arithexpression).
```

Usage: :- `calls AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is a call assertion body. (assertions_props:c_assrt_body/1)

calls/2: DECLARATION

This assertion is similar to a `calls/1` assertion but it is explicitely qualified with an assertion status. Non-qualified `calls/1` assertions are assumed to have `check` status.

Usage: :- `AssertionStatus calls AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is a call assertion body. (assertions_props:c_assrt_body/1)

success/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it only provides information about the answers to a predicate. The described answers might be conditioned to a particular way of calling the predicate.

For example, the following assertion specifies the answers of the `length/2` predicate *if* it is called as in the first mode of usage above (note that the previous pred assertion already conveys such information, however it also compelled the predicate calls, while the success assertion does not):

```
:- success length(L,N) : list * var => list * integer.
```

Usage: :- `success AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is a predicate assertion body. (assertions_props:s_assrt_body/1)

success/2: DECLARATION

success assertion This assertion is similar to a `success/1` assertion but it is explicitely qualified with an assertion status. The status of non-qualified `success/1` assertions is assumed to be `check`.

Usage: :- `AssertionStatus success AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is a predicate assertion body. (assertions_props:s_assrt_body/1)

test/1: DECLARATION

This assertion is similar to a success assertion but it specifies a concrete test case to be run in order verify (partially) that the predicate is working as expected. For example, the following test will verify that the length predicate works well for the particular list given:

```
:- test length(L,N) : ( L = [1,2,5,2] ) => ( N = 4 ).
```

Usage: :- `test AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is a predicate assertion body. (assertions_props:s_assrt_body/1)

test/2: DECLARATION

This assertion is similar to a `test/1` assertion but it is explicitely qualified with an assertion status. Non-qualified `test/1` assertions are assumed to have `check` status. In this context, check means that the test should be executed when the developer runs the test battery.

Usage: :- `AssertionStatus test AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is a predicate assertion body. (assertions_props:s_assrt_body/1)

comp/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it only provides information about the global execution properties of a predicate (note that such kind of information is also conveyed by pred assertions). The described properties might be conditioned to a particular way of calling the predicate.

For example, the following assertion specifies that the computation of `append/3` (see `lists`) will not fail *if* it is called as described (but does not compel the predicate to be called that way):

```
:- comp append(Xs,Ys,Zs) : var * var * var + not_fail.
```

Usage: :- `comp AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is a comp assertion body. (assertions_props:g_assrt_body/1)

comp/2: DECLARATION

This assertion is similar to a `comp/1` assertion but it is explicitely qualified. Non-qualified `comp/1` assertions are assumed the qualifier `check`.

Usage: :- `AssertionStatus comp AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is a comp assertion body. (assertions_props:g_assrt_body/1)

prop/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it flags that the predicate being documented is also a " property."

Properties are standard predicates, but which are *guaranteed to terminate for any possible instantiation state of their argument(s)*, do not perform side-effects which may interfere with the program behaviour, and do not further instantiate their arguments or add new constraints.

Provided the above holds, properties can thus be safely used as run-time checks. The program transformation used in `ciaopp` for run-time checking guarantees the third requirement. It also performs some basic checks on properties which in most cases are enough for the second requirement. However, it is the user's responsibility to guarantee termination of the properties defined. (See also Chapter 56 [Declaring regular types], page 355 for some considerations applicable to writing properties.)

The set of properties is thus a strict subset of the set of predicates. Note that properties can be used to describe characteristics of arguments in assertions and they can also be executed (called) as any other predicates.

Usage: :- `prop AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

prop/2: DECLARATION

This assertion is similar to a `prop/1` assertion but it is explicitely qualified. Non-qualified `prop/1` assertions are assumed the qualifier `check`.

Usage: :- `AssertionStatus prop AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

entry/1: DECLARATION

This assertion provides information about the *external* calls to a predicate. It is identical syntactically to a `calls/1` assertion. However, they describe only external calls, i.e., calls to the exported predicates of a module from outside the module, or calls to the predicates in a non-modular file from other files (or the user).

These assertions are *trusted* by the compiler. As a result, if their descriptions are erroneous they can introduce bugs in programs. Thus, `entry/1` assertions should be written with care.

An important use of these assertions is in providing information to the compiler which it may not be able to infer from the program. The main use is in providing information on the ways in which exported predicates of a module will be called from outside the module. This will greatly improve the precision of the analyzer, which otherwise has to assume that the arguments that exported predicates receive are any arbitrary term.

Usage: :- `entry AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is a call assertion body. (assertions_props:c_assrt_body/1)

exit/1: DECLARATION

This type of assertion provides information about the answers that an (exported) predicate provides for *external* calls. It is identical syntactically to a `success/1` assertion. However, it describes only external answers, i.e., answers to the exported predicates of a module from outside the module, or answers to the predicates in a non-modular file from other files (or the user). The described answers may be conditioned to a particular way of calling the predicate. E.g.:

```
:- exit length(L,N) : list * var => list * integer.
```

Usage: :- `exit AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is a predicate assertion body. (assertions_props:s_assrt_body/1)

exit/2: DECLARATION

exit assertion This assertion is similar to an `exit/1` assertion but it is explicitely qualified with an assertion status. Non-qualified `exit/1` assertions are assumed the qualifier `check`.

Usage: :- `AssertionStatus exit AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is a predicate assertion body. (assertions_props:s_assrt_body/1)

modedef/1: DECLARATION

This assertion is used to define modes. A mode defines in a compact way a set of call and success properties. Once defined, modes can be applied to predicate arguments in assertions. The meaning of this application is that the call and success properties defined by the mode hold for the argument to which the mode is applied. Thus, a mode is conceptually a "property macro".

The syntax of mode definitions is similar to that of pred declarations. For example, the following set of assertions:

```
:- modedef +A : nonvar(A) # "A is bound upon predicate entry.".

:- pred p(+A,B) : integer(A) =>  ground(B).
```

is equivalent to:

```
:- pred p(A,B) : (nonvar(A),integer(A)) =>  ground(B)
   # "A is bound upon predicate entry.".
```

Usage: :- `modedef AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

decl/1: DECLARATION

This assertion is similar to a `pred/1` assertion but it is used for declarations instead than for predicates.

Usage: :- `decl AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

decl/2: DECLARATION

This assertion is similar to a `decl/1` assertion but it is explicitely qualified. Non-qualified `decl/1` assertions are assumed the qualifier `check`.

Usage: :- `AssertionStatus decl AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

doc/2: DECLARATION

Usage: :- `doc(Pred,Comment).`

– *Description:* Documentation . This assertion provides a text `Comment` for a given predicate `Pred`.

– *The following properties should hold at call time:*

`Pred` is a head pattern. (assertions_props:head_pattern/1)

`Comment` is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments. (assertions_props:docstring/1)

comment/2: DECLARATION

Usage: :- `comment(Pred,Comment).`

– *Description:* An alias for `doc/2` (deprecated, for compatibility with older versions).

– *The following properties should hold at call time:*

`Pred` is a head pattern. (assertions_props:head_pattern/1)

`Comment` is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments. (assertions_props:docstring/1)

54.5 Documentation on exports (`assertions_doc`)

check/1: PREDICATE

Usage: `check(PropertyConjunction)`

– *Description:* This assertion provides information on a clause program point (position in the body of a clause). Calls to a `check/1` assertion can appear in the body of a clause in any place where a literal can normally appear. The property defined by `PropertyConjunction` should hold in all the run-time stores corresponding to that program point. See also Chapter 60 [Run-time checking of assertions], page 377.

– *The following properties should hold at call time:*

`PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (assertions_props:property_conjunction/1)

trust/1: PREDICATE

Usage: `trust(PropertyConjunction)`

– *Description:* This assertion also provides information on a clause program point. It is identical syntactically to a `check/1` assertion. However, the properties stated are not taken as something to be checked but are instead *trusted* by the compiler. While the compiler may in some cases detect an inconsistency between a `trust/1` assertion and the program, in all other cases the information given in the assertion will be taken to be true. As a result, if these assertions are erroneous they can introduce bugs in programs. Thus, `trust/1` assertions should be written with care.

An important use of these assertions is in providing information to the compiler which it may not be able to infer from the program (either because the information is not present or because the analyzer being used is not precise enough). In particular, providing information on external predicates which may not be accessible at the time of compiling the module can greatly improve the precision of the analyzer. This can be easily done with trust assertion.

– *The following properties should hold at call time:*

`PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (assertions_props:property_conjunction/1)

true/1: PREDICATE

Usage: `true(PropertyConjunction)`

– *Description:* This assertion is identical syntactically to a `check/1` assertion. However, the properties stated have been proved to hold by the analyzer. Thus, these assertions often represent the analyzer output.

– *The following properties should hold at call time:*

`PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (assertions_props:property_conjunction/1)

false/1: PREDICATE

Usage: `false(PropertyConjunction)`

– *Description:* This assertion is identical syntactically to a `check/1` assertion. However, the properties stated have been proved not to hold by the analyzer. Thus, these assertions often represent the analyzer output.

– *The following properties should hold at call time:*

`PropertyConjunction` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument. (assertions_props:property_conjunction/1)

55 Types and properties related to assertions

Author(s): Manuel Hermenegildo.

This module is part of the `assertions` library. It provides the formal definition of the syntax of several forms of assertions and describes their meaning. It does so by defining types and properties related to the assertions themselves. The text describes, for example, the overall fields which are admissible in the bodies of assertions, where properties can be used inside these bodies, how to combine properties for a given predicate argument (e.g., conjunctions) , etc. and provides some examples.

55.1 Usage and interface (`assertions_props`)

- **Library usage:**
  `:- use_module(library(assertions_props)).`
- **Exports:**
  - *Properties:*
    `head_pattern/1`, `nabody/1`, `docstring/1`.
  - *Regular Types:*
    `assrt_body/1`, `complex_arg_property/1`, `property_conjunction/1`, `property_starterm/1`, `complex_goal_property/1`, `dictionary/1`, `c_assrt_body/1`, `s_assrt_body/1`, `g_assrt_body/1`, `assrt_status/1`, `assrt_type/1`, `predfunctor/1`, `propfunctor/1`.

55.2 Documentation on exports (`assertions_props`)

assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `pred/1`, `decl/1`, etc. assertions. Such a body is of the form:

```
Pr [:: DP] [: CP] [=> AP] [+ GP] [# CO]
```

where (fields between [...] are optional):

- `Pr` is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- `DP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which expresses properties which are compatible with the predicate, i.e., instantiations made by the predicate are *compatible* with the properties in the sense that applying the property at any point would not make it fail.
- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `AP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *answers* to the predicate (if the predicate succeeds). These only apply if the (possibly empty) properties given for calls in the assertion hold.
- `GP` is a (possibly empty) complex goal property (`complex_goal_property/1`) which applies to the *whole execution* of a call to the predicate. These only apply if the (possibly empty) properties given for calls in the assertion hold.

- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

  See the `lpdoc` manual for documentation on assertion comments.

Usage: `assrt_body(X)`

– *Description:* `X` is an assertion body.

head_pattern/1: PROPERTY

A head pattern can be a predicate name (functor/arity) (`predname/1`) or a term. Thus, both `p/3` and `p(A,B,C)` are valid head patterns. In the case in which the head pattern is a term, each argument of such a term can be:

- A variable. This is useful in order to be able to refer to the corresponding argument positions by name within properties and in comments. Thus, `p(Input,Parameter,Output)` is a valid head pattern.
- A variable, as above, but preceded by a " mode." This mode determines in a compact way certain call or answer properties. For example, the head pattern `p(Input,+Parameter,Output)` is valid, as long as `+/1` is declared as a mode.

  Acceptable modes are documented in `library(basicmodes)` and `library(isomodes)`. User defined modes are documented in `modedef/1`.
- Any term. In this case this term determines the instantiation state of the corresponding argument position of the predicate calls to which the assertion applies.
- A ground term preceded by a " mode." The ground term determines a property of the corresponding argument. The mode determines if it applies to the calls and/or the successes. The actual property referred to is that given by the term but with one more argument added at the beginning, which is a new variable which, in a rewriting of the head pattern, appears at the argument position occupied by the term. For example, the head pattern `p(Input,+list(int),Output)` is valid for mode `+/1` defined in `library(isomodes)`, and equivalent in this case to having the head pattern `p(Input,A,Output)` and stating that the property `list(A,int)` holds for the calls of the predicate.
- Any term preceded by a " mode." In this case, only one variable is admitted, it has to be the first argument of the mode, and it represents the argument position. I.e., it plays the role of the new variable mentioned above. Thus, no rewriting of the head pattern is performed in this case. For example, the head pattern `p(Input,+(Parameter,list(int)),Output)` is valid for mode `+/2` defined in `library(isomodes)`, and equivalent in this case to having the head pattern `p(Input,Parameter,Output)` and stating that the property `list(Parameter,int)` holds for the calls of the predicate.

Usage: `head_pattern(Pr)`

– *Description:* `Pr` is a head pattern.

complex_arg_property/1: REGTYPE

`complex_arg_property(Props)`

`Props` is a (possibly empty) complex argument property. Such properties can appear in two formats, which are defined by `property_conjunction/1` and `property_starterm/1`

respectively. The two formats can be mixed provided they are not in the same field of an assertion. I.e., the following is a valid assertion:

```
:- pred foo(X,Y) : nonvar * var => (ground(X),ground(Y)).
```

Usage: `complex_arg_property(Props)`

– *Description:* `Props` is a (possibly empty) complex argument property

property_conjunction/1: REGTYPE

This type defines the first, unabridged format in which properties can be expressed in the bodies of assertions. It is essentially a conjunction of properties which refer to variables. The following is an example of a complex property in this format:

- `(integer(X),list(Y,integer))`: `X` has the property `integer/1` and `Y` has the property `list/2`, with second argument `integer`.

Usage: `property_conjunction(Props)`

– *Description:* `Props` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. The first argument of each such term is a variable which appears as a head argument.

property_starterm/1: REGTYPE

This type defines a second, compact format in which properties can be expressed in the bodies of assertions. A `property_starterm/1` is a term whose main functor is `*/2` and, when it appears in an assertion, the number of terms joined by `*/2` is exactly the arity of the predicate it refers to. A similar series of properties as in `property_conjunction/1` appears, but the arity of each property is one less: the argument position to which they refer (first argument) is left out and determined by the position of the property in the `property_starterm/1`. The idea is that each element of the `*/2` term corresponds to a head argument position. Several properties can be assigned to each argument position by grouping them in curly brackets. The following is an example of a complex property in this format:

- `integer * list(integer)`: the first argument of the procedure (or function, or ...) has the property `integer/1` and the second one has the property `list/2`, with second argument `integer`.
- `{integer,var} * list(integer)`: the first argument of the procedure (or function, or ...) has the properties `integer/1` and `var/1` and the second one has the property `list/2`, with second argument `integer`.

Usage: `property_starterm(Props)`

– *Description:* `Props` is either a term or several terms separated by `*/2`. The main functor of each of those terms corresponds to that of the definition of a property, and the arity should be one less than in the definition of such property. All arguments of each such term are ground.

complex_goal_property/1: REGTYPE

`complex_goal_property(Props)`

`Props` is a (possibly empty) complex goal property. Such properties can be either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. Such properties apply to all executions of all goals of the predicate which comply with the assertion in which the `Props` appear.

The arguments of the terms in `Props` are implicitely augmented with a first argument which corresponds to a goal of the predicate of the assertion in which the `Props` appear. For example, the assertion

```
:- comp var(A) + not_further_inst(A).
```

has property `not_further_inst/1` as goal property, and establishes that in all executions of `var(A)` it should hold that `not_further_inst(var(A),A)`.

Usage: `complex_goal_property(Props)`

– *Description:* `Props` is either a term or a *conjunction* of terms. The main functor and arity of each of those terms corresponds to the definition of a property. A first implicit argument in such terms identifies goals to which the properties apply.

nabody/1: PROPERTY

Usage: `nabody(ABody)`

– *Description:* `ABody` is a normalized assertion body.

dictionary/1: REGTYPE

Usage: `dictionary(D)`

– *Description:* `D` is a dictionary of variable names.

c_assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `call/1`, `entry/1`, etc. assertions. The following are admissible:

```
Pr : CP [# CO]
```

where (fields between [...] are optional):

- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

The format of the different parts of the assertion body are given by `n_assrt_body/5` and its auxiliary types.

Usage: `c_assrt_body(X)`

– *Description:* `X` is a call assertion body.

s_assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `pred/1`, `func/1`, etc. assertions. The following are admissible:

```
Pr : CP => AP # CO
Pr : CP => AP
Pr => AP # CO
Pr => AP
```

where:

- `Pr` is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `AP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *answers* to the predicate (if the predicate succeeds). These only apply if the (possibly empty) properties given for calls in the assertion hold.
- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

The format of the different parts of the assertion body are given by `n_assrt_body/5` and its auxiliary types.

Usage: `s_assrt_body(X)`

– *Description:* `X` is a predicate assertion body.

g_assrt_body/1: REGTYPE

This predicate defines the different types of syntax admissible in the bodies of `comp/1` assertions. The following are admissible:

```
Pr : CP + GP # CO
Pr : CP + GP
Pr + GP # CO
Pr + GP
```

where:

- `Pr` is a head pattern (`head_pattern/1`) which describes the predicate or property and possibly gives some implicit call/answer information.
- `CP` is a (possibly empty) complex argument property (`complex_arg_property/1`) which applies to the *calls* to the predicate.
- `GP` contains (possibly empty) complex goal property (`complex_goal_property/1`) which applies to the *whole execution* of a call to the predicate. These only apply if the (possibly empty) properties given for calls in the assertion hold.
- `CO` is a comment string (`docstring/1`). This comment only applies if the (possibly empty) properties given for calls in the assertion hold. The usual formatting commands that are applicable in comment strings can be used (see `stringcommand/1`).

The format of the different parts of the assertion body are given by `n_assrt_body/5` and its auxiliary types.

Usage: `g_assrt_body(X)`

– *Description:* `X` is a comp assertion body.

assrt_status/1: REGTYPE

The types of assertion status. They have the same meaning as the program-point assertions, and are as follows:

```
assrt_status(true).
assrt_status(false).
assrt_status(check).
assrt_status(checked).
```

```
assrt_status(trust).
```

Usage: `assrt_status(X)`

– *Description:* `X` is an acceptable status for an assertion.

assrt_type/1: REGTYPE

The admissible kinds of assertions:

```
assrt_type(pred).
assrt_type(prop).
assrt_type(decl).
assrt_type(func).
assrt_type(calls).
assrt_type(success).
assrt_type(comp).
assrt_type(entry).
assrt_type(exit).
assrt_type(test).
assrt_type(texec).
assrt_type(modedef).
```

Usage: `assrt_type(X)`

– *Description:* `X` is an admissible kind of assertion.

predfunctor/1: REGTYPE

Usage: `predfunctor(X)`

– *Description:* `X` is a type of assertion which defines a predicate.

propfunctor/1: REGTYPE

Usage: `propfunctor(X)`

– *Description:* `X` is a type of assertion which defines a *property*.

docstring/1: PROPERTY

Usage: `docstring(String)`

– *Description:* `String` is a text comment with admissible documentation commands. The usual formatting commands that are applicable in comment strings are defined by `stringcommand/1`. See the `lpdoc` manual for documentation on comments.

56 Declaring regular types

Author(s): Manuel Hermenegildo, Pedro López, Francisco Bueno.

This library package adds declarations and new operator definitions which provide simple syntactic sugar to write regular type definitions in source code. Regular types are just properties which have the additional characteristic of being regular types (`basic_props:regtype/1`), defined below.

For example, this library package allows writing:

```
:- regtype tree(X) # "X is a tree.".
```

instead of the more cumbersome:

```
:- prop tree(X) + regtype # "X is a tree.".
```

Regular types can be used as properties to describe predicates and play an essential role in program debugging (see the Ciao Prolog preprocessor (`ciaopp`) manual).

In this chapter we explain some general considerations worth taking into account when writing properties in general, not just regular types.

56.1 Defining properties

Given the classes of assertions in the Ciao assertion language, there are two fundamental classes of properties. Properties used in assertions which refer to execution states (i.e., `calls/1`, `success/1`, and the like) are called *properties of execution states*. Properties used in assertions related to computations (i.e., `comp/1`) are called *properties of computations*. Different considerations apply when writing a property of the former or of the later kind.

Consider a definition of the predicate `string_concat/3` which concatenates two character strings (represented as lists of ASCII codes):

```
string_concat([],L,L).
string_concat([X|Xs],L,[X|NL]):- string_concat(Xs,L,NL).
```

Assume that we would like to state in an assertion that each argument "is a list of integers." However, we must decide which one of the following two possibilities we mean exactly: "the argument is *instantiated* to a list of integers" (let us call this property `instantiated_to_intlist/1`), or "if any part of the argument is instantiated, this instantiation must be compatible with the argument being a list of integers" (we will call this property `compatible_with_intlist/1`). For example, `instantiated_to_intlist/1` should be true for the terms `[]` and `[1,2]`, but should not for `X`, `[a,2]`, and `[X,2]`. In turn, `compatible_with_intlist/1` should be true for `[]`, `X`, `[1,2]`, and `[X,2]`, but should not be for `[X|1]`, `[a,2]`, and `1`. We refer to properties such as `instantiated_to_intlist/1` above as *instantiation properties* and to those such as `compatible_with_intlist/1` as *compatibility properties* (corresponding to the traditional notions of "instantiation types" and "compatibility types").

It turns out that both of these notions are quite useful in practice. In the example above, we probably would like to use `compatible_with_intlist/1` to state that on success of `string_concat/3` all three argument must be compatible with lists of integers in an assertion like:

```
:- success string_concat(A,B,C) => ( compatible_with_intlist(A),
                                      compatible_with_intlist(B),
                                      compatible_with_intlist(C) ).
```

With this assertion, no error will be flagged for a call to `string_concat/3` such as `string_concat([20],L,R)`, which on success produces the resulting atom `string_concat([20],L,[20|L])`, but a call `string_concat([],a,R)` would indeed flag an error.

On the other hand, and assuming that we are running on a Prolog system, we would probably like to use `instantiated_to_intlist/1` for `sumlist/2` as follows:

```
:- calls sumlist(L,N) : instantiated_to_intlist(L).

sumlist([],0).
sumlist([X|R],S) :- sumlist(R,PS), S is PS+X.
```

to describe the type of calls for which the program has been designed, i.e., those in which the first argument of `sumlist/2` is indeed a list of integers.

The property `instantiated_to_intlist/1` might be written as in the following (Prolog) definition:

```
:- prop instantiated_to_intlist/1.

instantiated_to_intlist(X) :-
       nonvar(X), instantiated_to_intlist_aux(X).

instantiated_to_intlist_aux([]).
instantiated_to_intlist_aux([X|T]) :-
       integer(X), instantiated_to_intlist(T).
```

(Recall that the Prolog builtin `integer/1` itself implements an instantiation check, failing if called with a variable as the argument.)

The property `compatible_with_intlist/1` might in turn be written as follows (also in Prolog):

```
:- prop compatible_with_intlist/1.

compatible_with_intlist(X) :- var(X).
compatible_with_intlist(X) :-
       nonvar(X), compatible_with_intlist_aux(X).

compatible_with_intlist_aux([]).
compatible_with_intlist_aux([X|T]) :-
       int_compat(X), compatible_with_intlist(T).

int_compat(X) :- var(X).
int_compat(X) :- nonvar(X), integer(X).
```

Note that these predicates meet the criteria for being properties and thus the `prop/1` declaration is correct.

Ensuring that a property meets the criteria for "not affecting the computation" can sometimes make its coding somewhat tedious. In some ways, one would like to be able to write simply:

```
intlist([]).
intlist([X|R]) :- int(X), intlist(R).
```

(Incidentally, note that the above definition, provided that it suits the requirements for being a property and that `int/1` is a regular type, meets the criteria for being a regular type. Thus, it could be declared `:- regtype intlist/1`.)

But note that (independently of the definition of `int/1`) the definition above is not the correct instantiation check, since it would succeed for a call such as `intlist(X)`. In fact, it is not strictly correct as a compatibility property either, because, while it would fail or succeed as expected, it would perform instantiations (e.g., if called with `intlist(X)` it would bind `X` to `[]`). In practice, it is convenient to provide some run-time support to aid in this task.

The run-time support of the Ciao system (see Chapter 60 [Run-time checking of assertions], page 377) ensures that the execution of properties is performed in such a way that properties written as above can be used directly as instantiation checks. Thus, writing:

```
:- calls sumlist(L,N) : intlist(L).
```

has the desired effect. Also, the same properties can often be used as compatibility checks by writing them in the assertions as `compat(Property)` (`basic_props:compat/1`). Thus, writing:

```
:- success string_concat(A,B,C) => ( compat(intlist(A)),
                                     compat(intlist(B)),
                                     compat(intlist(C)) ).
```

also has the desired effect.

As a general rule, the properties that can be used directly for checking for compatibility should be *downwards closed*, i.e., once they hold they will keep on holding in every state accessible in forwards execution. There are certain predicates which are inherently *instantiation* checks and should not be used as *compatibility* properties nor appear in the definition of a property that is to be used with `compat`. Examples of such predicates (for Prolog) are `==`, `ground`, `nonvar`, `integer`, `atom`, `>`, etc. as they require a certain instantiation degree of their arguments in order to succeed.

In contrast with properties of execution states, *properties of computations* refer to the entire execution of the call(s) that the assertion relates to. One such property is, for example, `not_fail/1` (note that although it has been used as in `:- comp append(Xs,Ys,Zs) + not_fail`, it is in fact read as `not_fail(append(Xs,Ys,Zs))`; see `assertions_props:complex_goal_property/1`). For this property, which should be interpreted as "execution of the predicate either succeeds at least once or loops," we can use the following predicate `not_fail/1` for run-time checking:

```
not_fail(Goal):-
      if( call(Goal),
          true,            %% then
          warning(Goal) ). %% else
```

where the `warning/1` (library) predicate simply prints a warning message.

In this simple case, implementation of the predicate is not very difficult using the (non-standard) `if/3` builtin predicate present in many Prolog systems.

However, it is not so easy to code predicates which check other properties of the computation and we may in general need to program a meta-interpreter for this purpose.

56.2 Usage and interface (`regtypes_doc`)

- **Library usage:**
  `:- use_package(regtypes).`
  or
  `:- module(...,...,[regtypes]).`
- **New operators defined:**
  `regtype/1` [1150,fx], `regtype/2` [1150,xfx].
- **New declarations defined:**
  `regtype/1`, `regtype/2`.
- **Other modules used:**
  - *System library modules:*
    `assertions/assertions_props`.

56.3 Documentation on new declarations (regtypes_doc)

regtype/1: DECLARATION

This assertion is similar to a prop assertion but it flags that the property being documented is also a " regular type." Regular types are properties whose definitions are *regular programs* (see lelow). This allows for example checking whether it is in the class of types supported by the regular type checking and inference modules.

A regular program is defined by a set of clauses, each of the form:

```
p(x, v_1, ..., v_n)  :- body_1, ..., body_k.
```

where:

1. `x` is a term whose variables (which are called *term variables*) are unique, i.e., it is not allowed to introduce equality constraints between the variables of `x`.

   For example, `p(f(X, Y)) :- ...` is valid, but `p(f(X, X)) :- ...` is not.
2. in all clauses defining `p/n+1` the terms `x` do not unify except maybe for one single clause in which `x` is a variable.
3. `n >= 0` and `p/n` is a *parametric type functor* (whereas the predicate defined by the clauses is `p/n+1`).
4. `v_1`, ..., `v_n` are unique variables, which are called *parametric variables*.
5. Each `body_i` is of the form:
   1. `t(z)` where `z` is one of the *term variables* and `t` is a *regular type expression*;
   2. `q(y, t_1, ..., t_m)` where `m >= 0`, `q/m` is a *parametric type functor*, not in the set of functors `=/2`, `^/2`, `./3`.

      `t_1, ..., t_m` are *regular type expressions*, and `y` is a *term variable*.
6. Each term variable occurs at most once in the clause's body (and should be as the first argument of a literal).

A *regular type expression* is either a parametric variable or a parametric type functor applied to some of the parametric variables.

A parametric type functor is a regular type, defined by a regular program, or a basic type. Basic types are defined in Chapter 15 [Basic data types and properties], page 105.

The set of regular types is thus a well defined subset of the set of properties. Note that types can be used to describe characteristics of arguments in assertions and they can also be executed (called) as any other predicates.

Usage: :- `regtype AssertionBody`.

– *The following properties should hold at call time:*

`AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

regtype/2: DECLARATION

This assertion is similar to a `regtype/1` assertion but it is explicitely qualified. Non-qualified `regtype/1` assertions are assumed the qualifier `check`. Note that checking regular type definitions should be done with the `ciaopp` preprocessor.

Usage: :- `AssertionStatus regtype AssertionBody`.

– *The following properties should hold at call time:*

`AssertionStatus` is an acceptable status for an assertion. (assertions_props:assrt_status/1)

`AssertionBody` is an assertion body. (assertions_props:assrt_body/1)

57 Properties which are native to analyzers

Author(s): Francisco Bueno, Manuel Hermenegildo, Pedro López, Edison Mera.

This library contains a set of properties which are natively understood by the different program analyzers of `ciaopp`. They are used by `ciaopp` on output and they can also be used as properties in assertions.

57.1 Usage and interface (`native_props`)

- **Library usage:**
  `:- use_module(library(assertions(native_props)))`
  or also as a package `:- use_package(nativeprops)`.
  Note the different names of the library and the package.
- **Exports:**
  - *Properties:*
    `clique/1`,
    `clique_1/1`, `constraint/1`, `covered/1`, `covered/2`, `exception/1`, `exception/2`, `fails/1`, `finite_solutions/1`, `have_choicepoints/1`, `indep/1`, `indep/2`, `is_det/1`, `linear/1`, `mshare/1`, `mut_exclusive/1`, `no_choicepoints/1`, `no_exception/1`, `no_exception/2`, `no_signal/1`, `no_signal/2`, `non_det/1`, `nonground/1`, `not_covered/1`, `not_fails/1`, `not_mut_exclusive/1`, `num_solutions/2`, `solutions/2`, `possibly_fails/1`, `possibly_nondet/1`, `relations/2`, `sideff_hard/1`, `sideff_pure/1`, `sideff_soft/1`, `signal/1`, `signal/2`, `signals/2`, `size/2`, `size/3`, `size_lb/2`, `size_o/2`, `size_ub/2`, `size_metric/3`, `size_metric/4`, `steps/2`, `steps_lb/2`, `steps_o/2`, `steps_ub/2`, `tau/1`, `terminates/1`, `test_type/2`, `throws/2`, `user_output/2`.
- **Other modules used:**
  - *System library modules:*
    `terms_check`, `terms_vars`, `sort`, `lists`, `streams`, `file_utils`, `system`.

57.2 Documentation on exports (`native_props`)

clique/1: PROPERTY

`clique(X)`

`X` is a set of variables of interest, much the same as a sharing group but `X` represents all the sharing groups in the powerset of those variables. Similar to a sharing group, a clique is often translated to `ground/1`, `indep/1`, and `indep/2` properties.

Usage: `clique(X)`

- *Description:* The clique pattern is `X`.
- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP as `clique(X)`. (basic_props:native/2)

clique_1/1: PROPERTY

`clique_1(X)`

`X` is a set of variables of interest, much the same as a sharing group but `X` represents all the sharing groups in the powerset of those variables but disregarding the singletons. Similar to a sharing group, a clique_1 is often translated to `ground/1`, `indep/1`, and `indep/2` properties.

Usage: `clique_1(X)`

- *Description:* The 1-clique pattern is `X`.
- *The following properties should hold globally:*

  This predicate is understood natively by CiaoPP as `clique_1(X)`. (basic_props:native/2)

constraint/1: PROPERTY

`constraint(C)`

`C` contains a list of linear (in)equalities that relate variables and `int` values. For example, `[A < B + 4]` is a constraint while `[A < BC + 4]` or `[A = 3.4, B >= C]` are not.

Usage: `constraint(C)`

- *Description:* `C` is a list of linear equations
- *The following properties hold globally:*

  This predicate is understood natively by CiaoPP. (basic_props:native/1)

covered/1: PROPERTY

`covered(X)`

For any call of the form `X` there is at least one clause whose test succeeds (i.e., all the calls of the form `X` are covered) [DLGH97].

Usage: `covered(X)`

- *Description:* All the calls of the form `X` are covered.

covered/2: PROPERTY

`covered(X,Y)`

All variables occuring in `X` occur also in `Y`.

Usage: `covered(X,Y)`

- *Description:* `X` is covered by `Y`.
- *The following properties hold globally:*

  This predicate is understood natively by CiaoPP. (basic_props:native/1)

exception/1: PROPERTY

Meta-predicate with arguments: `exception(goal)`.

Usage: `exception(Goal)`

- *Description:* Calls of the form `Goal` throw an exception.

exception/2: PROPERTY

Meta-predicate with arguments: `exception(goal,?)`.

Usage: `exception(Goal,E)`

– *Description:* Calls of the form `Goal` throw an exception that unifies with `E`.

fails/1: PROPERTY

`fails(X)`

Calls of the form `X` fail.

Meta-predicate with arguments: `fails(goal)`.

Usage: `fails(X)`

– *Description:* Calls of the form `X` fail.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

finite_solutions/1: PROPERTY

`finite_solutions(X)`

Calls of the form `X` produce a finite number of solutions [DLGH97].

Meta-predicate with arguments: `finite_solutions(goal)`.

Usage: `finite_solutions(X)`

– *Description:* All the calls of the form `X` have a finite number of solutions.

have_choicepoints/1: PROPERTY

Meta-predicate with arguments: `have_choicepoints(goal)`.

Usage: `have_choicepoints(X)`

– *Description:* A call to `X` creates choicepoints.

indep/1: PROPERTY

Usage: `indep(X)`

– *Description:* The variables in pairs in `X` are pairwise independent.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP as `indep(X)`. (basic_props:native/2)

indep/2: PROPERTY

Usage: `indep(X,Y)`

– *Description:* `X` and `Y` do not have variables in common.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP as `indep([[X,Y]])`. (basic_props:native/2)

is_det/1: PROPERTY

`is_det(X)`

All calls of the form `X` are deterministic, i.e., produce at most one solution, or do not terminate. In other words, if `X` succeeds, it can only succeed once. It can still leave choice points after its execution, but when backtracking into these, it can only fail or go into an infinite loop.

Meta-predicate with arguments: `is_det(goal)`.

Usage: `is_det(X)`

- *Description:* All calls of the form `X` are deterministic.

linear/1: PROPERTY

`linear(X)`

`X` is bound to a term which is linear, i.e., if it contains any variables, such variables appear only once in the term. For example, `[1,2,3]` and `f(A,B)` are linear terms, while `f(A,A)` is not.

Usage: `linear(X)`

- *Description:* `X` is instantiated to a linear term.
- *The following properties hold globally:*

  This predicate is understood natively by CiaoPP. (basic_props:native/1)

mshare/1: PROPERTY

`mshare(X)`

`X` contains all *sharing sets* [JL88,MH89] which specify the possible variable occurrences in the terms to which the variables involved in the clause may be bound. Sharing sets are a compact way of representing groundness of variables and dependencies between variables. This representation is however generally difficult to read for humans. For this reason, this information is often translated to `ground/1`, `indep/1` and `indep/2` properties, which are easier to read.

Usage: `mshare(X)`

- *Description:* The sharing pattern is `X`.
- *The following properties should hold globally:*

  This predicate is understood natively by CiaoPP as `sharing(X)`. (basic_props:native/2)

mut_exclusive/1: PROPERTY

`mut_exclusive(X)`

For any call of the form `X` at most one clause succeeds, i.e., clauses are pairwise exclusive.

Meta-predicate with arguments: `mut_exclusive(goal)`.

Usage: `mut_exclusive(X)`

- *Description:* For any call of the form `X` at most one clause succeeds.

no_choicepoints/1: PROPERTY

Meta-predicate with arguments: `no_choicepoints(goal)`.

Usage: `no_choicepoints(X)`

– *Description:* A call to `X` does not create choicepoints.

no_exception/1: PROPERTY

Meta-predicate with arguments: `no_exception(goal)`.

Usage: `no_exception(Goal)`

– *Description:* Calls of the form `Goal` do not throw any exception.

no_exception/2: PROPERTY

Meta-predicate with arguments: `no_exception(goal,?)`.

Usage: `no_exception(Goal,E)`

– *Description:* Calls of the form `Goal` do not throw exception `E`.

no_signal/1: PROPERTY

Meta-predicate with arguments: `no_signal(goal)`.

Usage: `no_signal(Goal)`

– *Description:* Calls of the form `Goal` do not send any signal.

no_signal/2: PROPERTY

Meta-predicate with arguments: `no_signal(goal,?)`.

Usage: `no_signal(Goal,E)`

– *Description:* Calls of the form `Goal` do not send the signal `E`.

non_det/1: PROPERTY

`non_det(X)`

All calls of the form `X` are non-deterministic, i.e., produce several solutions.

Meta-predicate with arguments: `non_det(goal)`.

Usage: `non_det(X)`

– *Description:* All calls of the form `X` are non-deterministic.

nonground/1: PROPERTY

Usage: `nonground(X)`

– *Description:* `X` is not ground.

– *The following properties should hold globally:*

This predicate is understood natively by CiaoPP as `not_ground(X)`. (basic_props:native/2)

not_covered/1: PROPERTY

`not_covered(X)`

There is some call of the form `X` for which there is no clause whose test succeeds [DLGH97].

Usage: `not_covered(X)`

– *Description:* Not all of the calls of the form `X` are covered.

not_fails/1: PROPERTY

`not_fails(X)`

Calls of the form `X` produce at least one solution, or do not terminate [DLGH97].

Meta-predicate with arguments: `not_fails(goal)`.

Usage: `not_fails(X)`

– *Description:* All the calls of the form `X` do not fail.

– *The following properties hold globally:*
This predicate is understood natively by CiaoPP. (basic_props:native/1)

not_mut_exclusive/1: PROPERTY

`not_mut_exclusive(X)`

For calls of the form `X` more than one clause may succeed. I.e., clauses are not disjoint for some call.

Meta-predicate with arguments: `not_mut_exclusive(goal)`.

Usage: `not_mut_exclusive(X)`

– *Description:* For some calls of the form `X` more than one clause may succeed.

num_solutions/2: PROPERTY

Usage 1: `num_solutions(X,N)`

– *Description:* All the calls of the form `X` have `N` solutions.

– *If the following properties should hold at call time:*
`X` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
`N` is an integer. (basic_props:int/1)

Usage 2: `num_solutions(Goal,Check)`

– *Description:* For a call to `Goal`, `Check(X)` succeeds, where `X` is the number of solutions.

– *If the following properties should hold at call time:*
`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
`Check` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

solutions/2: PROPERTY

Usage: `solutions(Goal,Sols)`

– *Description:* Goal `Goal` produces the solutions listed in `Sols`.

– *If the following properties should hold at call time:*

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Sols` is a list. (basic_props:list/1)

possibly_fails/1: PROPERTY

`possibly_fails(X)`

Non-failure is not ensured for any call of the form `X` [DLGH97]. In other words, nothing can be ensured about non-failure nor termination of such calls.

Meta-predicate with arguments: `possibly_fails(goal)`.

Usage: `possibly_fails(X)`

– *Description:* Non-failure is not ensured for calls of the form `X`.

possibly_nondet/1: PROPERTY

`possibly_nondet(X)`

Non-determinism is not ensured for all calls of the form `X`. In other words, nothing can be ensured about determinacy nor termination of such calls.

Usage: `possibly_nondet(X)`

– *Description:* Non-determinism is not ensured for calls of the form `X`.

relations/2: PROPERTY

`relations(X,N)`

The goal `X` produces `N` solutions. In other words, `N` is the cardinality of the solution set of `X`.

Meta-predicate with arguments: `relations(goal,?)`.

Usage: `relations(X,N)`

– *Description:* Goal `X` produces `N` solutions.

sideff_hard/1: PROPERTY

Meta-predicate with arguments: `sideff_hard(goal)`.

Usage: `sideff_hard(X)`

– *Description:* `X` has *hard side-effects*, i.e., those that might affect program execution (e.g., assert/retract).

sideff_pure/1: PROPERTY

Meta-predicate with arguments: `sideff_pure(goal)`.

Usage: `sideff_pure(X)`

– *Description:* `X` is pure, i.e., has no side-effects.

sideff_soft/1: PROPERTY

Meta-predicate with arguments: `sideff_soft(goal)`.

Usage: `sideff_soft(X)`

– *Description:* `X` has *soft side-effects*, i.e., those not affecting program execution (e.g., input/output).

signal/1: PROPERTY

Meta-predicate with arguments: `signal(goal)`.

Usage: `signal(Goal)`

– *Description:* Calls of the form `Goal` throw a signal.

signal/2: PROPERTY

Meta-predicate with arguments: `signal(goal,?)`.

Usage: `signal(Goal,E)`

– *Description:* A call to `Goal` sends a signal that unifies with `E`.

signals/2: PROPERTY

Meta-predicate with arguments: `signals(goal,?)`.

Usage: `signals(Goal,Es)`

– *Description:* Calls of the form `Goal` can generate only the signals that unify with the terms listed in `Es`.

size/2: PROPERTY

Usage: `size(X,Y)`

– *Description:* `Y` is the size of argument `X`, for any approximation.

size/3: PROPERTY

Usage: `size(A,X,Y)`

– *Description:* `Y` is the size of argument `X`, for the approximation `A`.

size_lb/2: PROPERTY

`size_lb(X,Y)`

The minimum size of the terms to which the argument `Y` is bound is given by the expression `Y`. Various measures can be used to determine the size of an argument, e.g., list-length, term-size, term-depth, integer-value, etc. [DL93,LGHD96].

Usage: `size_lb(X,Y)`

– *Description:* `Y` is a lower bound on the size of argument `X`.

size_o/2: PROPERTY

Usage: `size_o(X,Y)`

– *Description:* The size of argument `X` is in the order of `Y`.

size_ub/2: PROPERTY

`size_ub(X,Y)`

The maximum size of the terms to which the argument `Y` is bound is given by the expression `Y`. Various measures can be used to determine the size of an argument, e.g., list-length, term-size, term-depth, integer-value, etc. [DL93,LGHD96].

Usage: `size_ub(X,Y)`

– *Description:* `Y` is a upper bound on the size of argument `X`.

size_metric/3: PROPERTY

Meta-predicate with arguments: `size_metric(goal,?,?)`.

Usage: `size_metric(Head,Var,Metric)`

– *Description:* `Metric` is the metric of the variable `Var`, for any approximation.

size_metric/4: PROPERTY

Meta-predicate with arguments: `size_metric(goal,?,?,?)`.

Usage: `size_metric(Head,Approx,Var,Metric)`

– *Description:* `Metric` is the metric of the variable `Var`, for the approximation `Approx`. Currently, `Metric` can be: `int/1`, `size/1`, `length/1`, `depth/2`, and `void/1`.

steps/2: PROPERTY

`steps(X,Y)`

The time (in resolution steps) spent by any call of the form `X` is given by the expression `Y`

Meta-predicate with arguments: `steps(goal,?)`.

Usage: `steps(X,Y)`

– *Description:* `Y` is the cost (number of resolution steps) of any call of the form `X`.

steps_lb/2: PROPERTY

`steps_lb(X,Y)`

The minimum computation time (in resolution steps) spent by any call of the form `X` is given by the expression `Y` [DLGHL97,LGHD96]

Meta-predicate with arguments: `steps_lb(goal,?)`.

Usage: `steps_lb(X,Y)`

– *Description:* `Y` is a lower bound on the cost of any call of the form `X`.

steps_o/2: PROPERTY

Meta-predicate with arguments: `steps_o(goal,?)`.

Usage: `steps_o(X,Y)`

– *Description:* `Y` is the complexity order of the cost of any call of the form `X`.

steps_ub/2: PROPERTY

`steps_ub(X,Y)`

The maximum computation time (in resolution steps) spent by any call of the form `X` is given by the expression `Y` [DL93,LGHD96].

Meta-predicate with arguments: `steps_ub(goal,?)`.

Usage: `steps_ub(X,Y)`

– *Description:* `Y` is a upper bound on the cost of any call of the form `X`.

tau/1: PROPERTY

`tau(Types)`

`Types` contains a list with the type associations for each variable, in the form `V/[T1,..,TN]`. Note that tau is used in object-oriented programs only

Usage: `tau(TypeInfo)`

– *Description:* `Types` is a list of associations between variables and list of types

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

terminates/1: PROPERTY

`terminates(X)`

Calls of the form `X` always terminate [DLGH97].

Meta-predicate with arguments: `terminates(goal)`.

Usage: `terminates(X)`

– *Description:* All calls of the form `X` terminate.

test_type/2: PROPERTY

Meta-predicate with arguments: `test_type(goal,?)`.

Usage: `test_type(X,T)`

– *Description:* Indicates the type of test that a predicate performs. Required by the nonfailure analyisis.

throws/2: PROPERTY

Meta-predicate with arguments: `throws(goal,?)`.

Usage: `throws(Goal,Es)`

– *Description:* Calls of the form `Goal` can throw only the exceptions that unify with the terms listed in `Es`.

user_output/2: PROPERTY

Meta-predicate with arguments: `user_output(goal,?)`.

Usage: `user_output(Goal,S)`

– *Description:* Calls of the form `Goal` write `S` to standard output.

instance/2: PROPERTY

Usage: `instance(Term1,Term2)`

– *Description:* `Term1` is an instance of `Term2`.

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP. (basic_props:native/1)

57.3 Known bugs and planned improvements (`native_props`)

- A missing property is succeeds (not_fails = succeeds or not_terminates. – EMM
- run-time check for throws/2 unimplemented yet.
- run-time check for signals/2 unimplemented yet.

58 ISO-Prolog modes

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This file defines the " modes" used in the documentation of the ISO-Prolog standard. See also Chapter 59 [Classical Prolog modes], page 373 for an alternative set of modes.

58.1 Usage and interface (`isomodes_doc`)

- **Library usage:**
  `:- use_package([assertions,isomodes]).`
- **New operators defined:**
  `?/1` [200,fy], `@/1` [200,fy].
- **New modes defined:**
  `+/1`, `@/1`, `-/1`, `?/1`, `*/1`, `+/2`, `@/2`, `-/2`, `?/2`, `*/2`.

58.2 Documentation on new modes (`isomodes_doc`)

+/1: MODE

Usage: `+A`

– *The following properties are added at call time:*
`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

@/1: MODE

Usage: `@A`

– *The following properties are added globally:*
`A` is not further instantiated. (basic_props:not_further_inst/2)

-/1: MODE

Usage: `-A`

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

?/1: MODE

Unspecified argument.

***/1:** MODE

Unspecified argument.

+/2: MODE

Usage: `A+X`

– *The following properties are added at call time:*
undefined:call(X,A) (undefined property)

@/2: MODE

Usage: `@(A,X)`

– *The following properties are added at call time:*
undefined:call(X,A) (undefined property)
– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)
– *The following properties are added globally:*
`A` is not further instantiated. (basic_props:not_further_inst/2)

-/2: MODE

Usage: `A-X`

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)
– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)

?/2: MODE

Usage: `A?X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)
– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)

***/2:** MODE

Usage: `A*X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

59 Classical Prolog modes

Author(s): Manuel Hermenegildo.

This file defines a number of very simple " modes" which are frequently useful in programs. These correspond to the modes used in classical Prolog texts with some simple addtions. Note that some of these modes use the same symbol as one of the ISO-modes (see Chapter 58 [ISO-Prolog modes], page 371) but with subtly different meaning.

59.1 Usage and interface (`basicmodes_doc`)

- **Library usage:**
  :- use_package([assertions,basicmodes]).
- **New operators defined:**
  `?/1` [500,fx], `@/1` [500,fx].
- **New modes defined:**
  `+/1`, `-/1`, `?/1`, `@/1`, `in/1`, `out/1`, `go/1`, `+/2`, `-/2`, `?/2`, `@/2`, `in/2`, `out/2`, `go/2`.
- **Other modules used:**
  - *System library modules:*
    `metaprops/meta_props`.

59.2 Documentation on new modes (`basicmodes_doc`)

+/1: MODE

Input value in argument.

Usage: `+A`

- *The following properties are added at call time:*
  `A` is currently a term which is not a free variable. (term_typing:nonvar/1)

-/1: MODE

No input value in argument.

Usage: `-A`

- *The following properties are added at call time:*
  `A` is a free variable. (term_typing:var/1)

?/1: MODE

Unspecified argument.

@/1: MODE

No output value in argument.

Usage: `@A`

- *The following properties are added globally:*
  `A` is not further instantiated. (basic_props:not_further_inst/2)

in/1: MODE

Input argument.

Usage: `in(A)`

– *The following properties are added at call time:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

out/1: MODE

Output argument.

Usage: `out(A)`

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

go/1: MODE

Ground output (input/output argument).

Usage: `go(A)`

– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

+/2: MODE

Usage: `A+X`

– *Call and exit are compatible with:*
`A` has property `X`. (meta_props:call/2)

– *The following properties are added at call time:*
`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

-/2: MODE

Usage: `A-X`

– *Call and exit are compatible with:*
`A` has property `X`. (meta_props:call/2)

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

?/2: MODE

Usage: `A?X`

– *Call and exit are compatible with:*
`A` has property `X`. (meta_props:call/2)

@/2: MODE

Usage: `@(A,X)`

– *Call and exit are compatible with:*
`A` has property `X`. (meta_props:call/2)
– *The following properties are added globally:*
`A` is not further instantiated. (basic_props:not_further_inst/2)

in/2: MODE

Usage: `in(A,X)`

– *Call and exit are compatible with:*
`A` has property `X`. (meta_props:call/2)
– *The following properties are added at call time:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)
– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

out/2: MODE

Usage: `out(A,X)`

– *Call and exit are compatible with:*
`A` has property `X`. (meta_props:call/2)
– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)
– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

go/2: MODE

Usage: `go(A,X)`

– *Call and exit are compatible with:*
`A` has property `X`. (meta_props:call/2)
– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

60 Run-time checking of assertions

Author(s): Edison Mera.

This package provides a complete implementation of run-time checks of predicate assertions. The program is instrumented to check such assertions at run time, and in case a property does not hold, the error is reported. Note that there is also an older package called rtchecks, by David Trallero. The advantage of this one is that it can be used independently of CiaoPP and also has updated functionality.

There are two main applications of run-time checks:

- To improve debugging of certain predicates, specifying some expected behavior that is checked at run-time with the assertions.
- To avoid manual implementation of run-time checks that should be done in some predicates, leaving the code clean and understandable.

The run-time checks can be configured using prolog flags. Below we itemize the valid prolog flags with its values and a brief explanation of the meaning:

- `rtchecks_level`
  - `exports`: Only use rtchecks for external calls of the exported predicates.
  - `inner` : Use also rtchecks for internal calls. Default.
- `rtchecks_trust`
  - `no` : Disable rtchecks for trust assertions.
  - `yes` : Enable rtchecks for trust assertions. Default.
- `rtchecks_entry`
  - `no` : Disable rtchecks for entry assertions.
  - `yes` : Enable rtchecks for entry assertions. Default.
- `rtchecks_exit`
  - `no` : Disable rtchecks for exit assertions.
  - `yes` : Enable rtchecks for exit assertions. Default.
- `rtchecks_test`
  - `no` : Disable rtchecks for test assertions. Default.
  - `yes` : Enable rtchecks for test assertions. Used for debugging purposes, but is better to use the unittest library.
- `rtchecks_inline`
  - `no` : Instrument rtchecks using call to library predicates present in `rtchecks_rt.pl`, `nativeprops.pl` and `basic_props.pl`. In this way, space is saved, but sacrifying performance due to usage of meta calls and external methods in the libraries. Default.
  - `yes` : Expand library predicates inline as far as possible. In this way, the code is faster, because its avoids metacalls and usage of external methods, but the final executable could be bigger.
- `rtchecks_asrloc` Controls the usage of locators for the assertions in the error messages. The locator says the file and lines that contains the assertion that had failed. Valid values are:
  - `no` : Disabled.
  - `yes` : Enabled. Default.
- `rtchecks_predloc` Controls the usage of locators for the predicate that caused the run-time check error. The locator says the first clause of the predicate that the violated assertion refers to.
  - `no` : Disabled.

    - `yes` : Enabled, Default.
- `rtchecks_callloc`
    - `no` : Do not show the stack of predicates that caused the failure
    - `predicate`: Show the stack of predicates that caused the failure. Instrument it in the predicate. Default.
    - `literal` : Show the stack of predicates that caused the failure. Instrument it in the literal. This mode provides more information, because reports also the literal in the body of the predicate.
- `rtchecks_namefmt`
    - `long` : Show the name of predicates, properties and the values of the variables
    - `short` : Only show the name of the predicate in a reduced format. Default.

60.1 Usage and interface (`rtchecks_doc`)

- **Library usage:**

  ```
  :- use_package(rtchecks).
  ```

  or

  ```
  :- module(...,...,[rtchecks]).
  ```

61 Unit Testing Library

Author(s): Edison Mera.

This library provides an extension of the Ciao assertion language which allows writing *unit tests*. The central idea is to use the assertion language to provide specifications of test cases for a given predicate. The package also provides some special properties that are convenient when specifying unit tests and the required run-time libraries.

In general, a *test assertion* is written as follows:

```
:- test predicate(A1, A2, ..., An)
   :  <Precondition>
   => <Postcondition>
   +  <Global properties>
   #  <Comment>.
```

Where the fields of the test assertion have the usual meaning in Ciao assertions, i.e., they contain conjunctions of properties which must hold at certain points in the execution. Here we give a somewhat more operational ("test oriented"), reading to these fields: `predicate/n` is the predicate to be tested. `Precondition` is a goal that is called before the predicate being tested, and can be used to generate values of the input parameters. `Postcondition` is a goal that should succeed after `predicate/n` has been called. The idea appears to be simple, but note that due to the non-determinism of logic programs, the test engine needs to test all the solutions that can be tested up to given limits (for example, a maximum number of solutions, or a given time-out). `Properties` specifies some global properties that the predicate should meet, for example, `not_fails` means that the program does not fail, `exception(error(a,b))` means that the program should throw the exception `error(a,b)`, and so on. But there are some specific properties that only applies to testing specified in the module unittest_props.pl, for example `times(N)` specifies that the given test should be executed N times, `try_sols(N)` specifies that the first N solutions of the predicate `predicate/n` are tested. `Comment` is a string that document the test.

A convenient way to run these tests is by selecting options in the CiaoDbg menu within the development environment:

1. `Run tests in current module`: execute only the tests specified in the current module.
2. `Run tests in all related modules`: execute the tests specified in the module and in all the modules being used by this.
3. `Show untested predicates`: show the *exported* predicates that do not have any test assertion.

61.1 Additional notes

1. The test assertions allow performing *unit* testing, i.e., in Ciao, performing tests *at the predicate level*.
2. The tests currently can only be applied to exported predicates.
3. If you need to write tests for predicates that are spread over several modules, but work together, then it is best to create a separate module, and reexport to the predicates required to build the test. This allows performing *integration testing*, using the same syntax of the unit tests.
4. The Ciao system includes a good (and growing) number of unit tests. To run all the tests among the other standard tests within the CiaoDE run the following (at the top level of the source tree):

   ```
   ./ciaosetup runtests
   ```

61.2 Usage and interface (unittest_doc)

- **Library usage:**
  `:- use_module(library(unittest)).`

PART VI - Ciao library miscellanea

Author(s): The CLIP Group.

This part documents several Ciao libraries which provide different useful additional functionality. Such functionality includes performing operating system calls, gathering statistics from the Ciao engine, file and filename manipulation, error and exception handling, fast reading and writing of terms (marshalling and unmarshalling), file locking, issuing program and error messages, pretty-printing programs and assertions, a browser of the system libraries, additional expansion utilities, concurrent aggregates, graph visualization, etc.

62 Library Paths for Ciao Components

Author(s): The CLIP Group.

This package setups the file search path and library aliases to access all the available Ciao components.

62.1 Usage and interface (`ciaopaths_doc`)

- **Library usage:**
  `:- use_package(ciaopaths).`
  or
  `:- module(...,...,[ciaopaths]).`
- **New operators defined:**
  `-->/2` [1200,xfx], `|/2` [1100,xfy].
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`.

62.2 Known bugs and planned improvements (`ciaopaths_doc`)

- The package is disabled by default for efficiency reasons (forces many dependencies on minimal programs). Nevertheless, a lighter implementation could be enabled by default for the 'ciao' dialect in the near future (and make the package disappear).

63 Analytic benchmarks

Author(s): Manuel Carro (adapted to Ciao Prolog).

This module provides a set of analytic benchmarks which try to isolate and measure the speed of certain very common operations in a Prolog system. The benchmarks come from a variety of sources, mostly within former ECRC (please look at the comments in the source code) and were posted by Jean-Claude Syre on 24th June 1986 to the Prolog Digest. Further packaging was done by Thoma Sjoland and SICS. They were adapted to Ciao Prolog by Manuel Carro, by:

- Changing the syntax to Ciao Prolog, when needed, including extensive use of higher-order programming for the benchmarking loops.
- Adapting the size of the benchmarks and the number of repetitions to fit modern computers and Prolog compilation.
- Changing the format of the output and recording performance data in the incore database.
- Changing benchmarking loops to be failure-driven.
- Adding a void initial dry run to heat up the caches.

The comments corresponding to the **original** programs follow. They have been largely unchanged, except to reflect changes in the program interface necessary to perform the modularization and to adapt them to Ciao Prolog. Of course the number of repeated calls was changed. The original comments are still in the source files.

63.1 Testing Calls

This is the one you always dreamed to test! Like all benchmarks, it uses a loop calling the actual benchmark program. The benchmark program consists of a sequence of 200 predicates having no arguments, no choice points, NOTHING. 200 is chosen to have sufficient accuracy in measuring the execution time.

The results show the effect of pure calls, and the Klips performance can be called the peak performance of the prolog system. Note that the peak performance has very little significance to classify the overall performance of a Prolog system.

63.2 Testing non-deterministic behavior

This program contains a series of 3 different benchmark predicates.

The predicate `choice_point/1` tests calls invoking the creation of a choice point, i.e. a branch point where the execution will possibly come back to in case of backtracking. It does NOT backtrack. Two versions are proposed, one with and the other without arguments.

We then present two predicates to evaluate the mechanism of backtracking during execution. Both predicates create one choice_point and then backtrack 20 times on every loop iteration step. `baktrak1/1` exhibits a kind of backtracking called *deep*, while `baktrak2/1` deals with *shallow* backtracking. Both are worth being tried, whatever your particular Prolog System is.

63.3 Testing environment handling

The creation and deletion of environments are an important feature in prolog machines. The following program attempts to evaluate that. A usual condition to have environments is that the clause is made of several goals. Thus there will be calls in the clause creating environments, and some work to set the parameters of each call. Three arguments per goal were chosen because this number comes close to the average number of arguments of a predicate and to the average number of permanent variables in an environment. The arguments were arranged in different orders for every goal, because we did not want to measure the merits of register transfer

optimisations. Note that these transfers exist, so the results cannot be compared with those given by the program which creates choice points (generally slower).

Another version, `envir0ar/1`, with 0 argument in each call, can also be usefully tried

63.4 Testing indexing mechanisms

We give only one test for indexing, i.e. the selection of a clause due to the type of an argument. This program does not test the merits of indexing on an argument other than the first one. It does not test for multiple indexing either. It does not show the inefficiency which occurs if 2 choice points per clause are created. This may happen e.g. in Warren's indexing scheme.

Each of these tests would require an extra benchmark program. The program given below tests the main point in indexing. Right now we think it is not worth adding all this complexity to the benchmarks, in order to measure all the details in indexing. Therefore we give only this single test.

63.5 Testing unification

We have 6 programs to evaluate the process of unification in the Prolog system:

- Test of list construction via unification.
- Test of list matching unification.
- Test of structure construction via unification This program is equivalent to construct_list, except that it uses the standard structure representation instead of the simplified list notation.
- Test of structure matching via unification. This predicate matches a list of 100 elements in structure notation.
- Test to match a nested structure. This predicate tests the (compiled) unification of a complex structure.
- Test of general unification of 2 complex structures. This predicate tests general unification. We call it general unification, because it cannot be analysed at compile time. Therefore this kind of unification cannot be compiled and, even in a compiled system, it must be handled at run time, exactly as by an interpreter. This is done by a general procedure for unification. The name of the benchmark therefore does not reflect that the unification is general, i.e. including all Prolog types (e.g. it does not contain variables), but it reflects the use of the procedure for general unification as opposed to specific, compiled unification.

Manuel Carro: note that in this case the term "Logical Inference" is a bit contrived, since by design some of these (head) unifications are very more compled, naturally being slower and giving slow KLIPS results.

63.6 Testing dereferencing

Program to benchmark the dereferencing speed. It constructs a list containing 500 variables which are then bound together. Since different systems use different strategies for binding variables on the global stack, the whole is made for two lists and the long variable chain is created only in one of them.

Manuel Carro: different results in this benchmark are not likely to affect larger, general programs. It is a well-known fact that n programs tend not to generate long dereferencing chains. Empirical measurements show that dereference chains of length greater than three are extremely rare. So a suboptimal / optimal behavior in this test is not likely to affect greatly the overall speed of a system.

63.7 Testing the cut

It seems almost impossible to isolate the cut operator in a simple test program. However, the cut-testing program in this benchmark set contains a lot of cut at exec time. It may be regarded as a partial test of cut, and may be worthwhile for some software implementations of Prolog. `cuttest/1` calls the cutit11 predicate, which performs 100 calls to a predicate cutt1 where a cut operator appears in the second clause. Having indexing makes the evaluation of the cut more accurate, so please indicate in our result whether or not your Prolog system uses indexing, to clarify the comparison with others.

63.8 Assorted small programs

Here we deal with prolog programs that do something, while being still small but representative of some well-known Prolog computations. This set should be augmented by other programs, some of them might come from your ideas.

Some of the following programs were taken from the Berkeley paper by Peter Van Roy "A Prolog Compiler for the PLM". Other programs were kindly donated by the following ECRC members: Helmut Simonis, Mehmet Dincbas, Micha Meier and Pascal Van Hentenryck.

The programs have been statically analysed and they represent fairly standard programs as far as the statistical averages are concerned. That is the arity of most clauses is 2 or 3 and there are usually 2 or 3 clauses per predicate. The programs range from fairly trivial programs like fibonacci series to problems such as Hamiltonian graph traversal.

Also, some more programs have been added since the last release and some corrections have been made. Most of the writes were removed in order to reduce i/o activity.

The programs added were symbolic differentiation (from Warren's paper) and a quick sort algorithm using difference lists. The last addition is a bit of a rogue: its a naive reverse, where one can enter the list length. The list gets constructed and then gets reversed.

We are grateful to Saumya Debray from Stony Brook and others for comments, suggestions, feedback and useful inputs.

These benchmarks were run on a VAX 785 with 8 Meg of memory, under 4.2 BSD Unix. The interpreter was C-Prolog version 1.5.

This entire file (without mail/net headers) contains 584 lines.

Name	Call by	# of Inferences (one iteration)	KLips (C-Prolog)
fib	fibonacci(1).	4932	2.0
map	map(200).	68	1.3
mham	mham(1).	493824	1.7
mutest	mutest(1).	1366	2.3
quicksort	qs(10).	601	1.9
queens	qu(10).	684	1.7
query	query(1).	2294	0.9
sym_diff	differen(150).	71	1.5

```
diff_lists| diff(50).          |          608       |   2.1
----------+-------------------+--------------------+----------
nrev  10  | nrev.              |           66       |   2.0
----------+-------------------+--------------------+----------
nrev  30  | nrev.              |          496       |   2.5
----------+-------------------+--------------------+----------
nrev  50  | nrev.              |         1326       |   2.5
----------+-------------------+--------------------+----------
nrev 100  | nrev.              |         5151       |   2.5
----------+-------------------+--------------------+----------
nrev 150  | nrev.              |        11476       |   2.5
----------+-------------------+--------------------+----------
nrev 200  | nrev.              |        20301       |   2.5
----------+-------------------+--------------------+----------
```

63.9 Usage and interface (ecrc)

- **Library usage:**
  `:- use_module(library(ecrc)).`
- **Exports:**
  - *Predicates:*
    `main/1`, `just_benchmarks/0`, `generate_human_file/0`, `generate_machine_file/0`, `send_info_to_developers/0`, `arithm_average/2`, `geom_average/2`.
  - *Regular Types:*
    `benchmark_usage/1`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `format`, `lists`, `system`, `dec10_io`, `terms`, `hiordlib`, `getopts`, `prolog_sys`, `benchmarks/benchmark_utilities`, `benchmarks/boresea`, `benchmarks/choice`, `benchmarks/envir`, `benchmarks/index`, `benchmarks/unif`, `benchmarks/deref`, `benchmarks/cut`, `benchmarks/small_programs`, `benchmarks/results`.

63.10 Documentation on exports (ecrc)

main/1: PREDICATE

Usage: `main(Flags)`

- *Description:* Main entry point. Execute all benchmarks and report on the performance obtained. This makes it easy to run the set of benchmarks as an executable. Its behavior regarding printing gathered data can be controlled with the list of flags passed as argument. Data is **always** asserted and available to other programs through the `dump_benchmark_data/0` and `access_benchmark_data/8` predicates.
- *The following properties should hold at call time:*
  `Flags` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Flags` is a list of `benchmark_usage`s. (basic_props:list/2)

benchmark_usage/1: REGTYPE

Usage: `benchmark_usage(Flag)`

– *Description:* Options which determine what this module should do with the execution results when called through the `main/1` entry point (i.e., if compiled to an executable). It is defined as

```
benchmark_usage('--estimation').
benchmark_usage('--no-machine').
benchmark_usage('--no-human').
benchmark_usage('--send-info').
benchmark_usage('--base-file-name').
```

with the following meaning:

- `'--no-human'`: do **not** dump human-readable data.
- `'--no-machine'`: do **not** dump data as a series of facts (which is a machine-readable format) which can be saved to a file and later read back in Prolog.
- `'--send-info'`: send a mail to the Ciao developers with the information gathered plus a terse description of the machine (O.S., architecture, CPU type and speed). The existence of a suitable user command to send mail is expected. No message is sent otherwise. No sensible (personal, etc.) information is gathered or sent.
- `--base-file-name` *file-name*: use *file-name* as a base to generate file with the reports this module generates. The machine-oriented file will have the `.pl` extension and the human-oriented file will have the `.txt` extension.

The options aboce can be used when calling `main/1` predicate or as command-line options for an executable built from this file. Note that the default options **print** available data both in human-readable and machine-readable formats.

– *Call and exit should be compatible with:*

`Flag` is an atom. (basic_props:atm/1)

just_benchmarks/0: PREDICATE

Usage:

– *Description:* Run the set of benchmarks in this program and save the speed information gathered. They can be later accessed using the predicates `generate_machine_file/0` or `generate_human_file/0`.

generate_human_file/0: PREDICATE

Usage:

– *Description:* Print to standard output a human-readable report of the information gathered by running `just_benchmarks/0`.

generate_machine_file/0: PREDICATE

Usage:

– *Description:* Print to standard output a machine-readable report of the information gathered by running `just_benchmarks/0`.

send_info_to_developers/0: PREDICATE

Usage:

– *Description:* Send a message to the Ciao developers with a report of the information gathered by running `just_benchmarks/0`.

arithm_average/2: PREDICATE

No further documentation available for this predicate.

geom_average/2: PREDICATE

No further documentation available for this predicate.

63.11 Known bugs and planned improvements (ecrc)

- The actual logical inferences each benchmark does has to be checked.

64 Parse and return command-line options

Author(s): Manuel Carro.

64.1 Usage and interface (getopts)

- **Library usage:**
  `:- use_module(library(getopts)).`
- **Exports:**
  - *Predicates:*
    `getopts/4`, `cl_option/2`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

64.2 Documentation on exports (getopts)

getopts/4: PREDICATE

Usage: `getopts(Arguments,Opts,Matched,Rest)`

– *Description:* Ciao Prolog parses the command-line arguments of its executables and passes them as a list of atoms to the `main/1` predicate. Thus, a shell invocation such as

```
./my_program -file input.txt -file input2.txt --output_file out.txt
-create-dir --decode --unsorte
```

makes `main/1` receive an argument such as

```
['-file', 'input.txt', '-file', 'input2.txt', '--output_file',
'out.txt', '-create-dir', '--decode', '--unsorte']
```

`getopts/4` can be used to parse such an command-line option list. passed in the `Arguments` parameter. `Opts` is a list of expected options, each option being an *option spec*, i.e., a term of the form `atom/arity`. For every `atom` a command-line option of the form `'--atom'` or `'-atom'` is expected, with `arity` arguments following it. An `arity` of zero can be omitted. For each matched option spec, the list `Matched` will contain a term of the form `atom(Arg1, Arg2, ..., Argn)`, where `n = arity`. The list `Rest` will contain the unmatched element in `Arguments`.

`Rest` will respect the relative order of the elements in `Arguments`. The matching elements in `Matched` appear in the same order as the options in `Opts`, and for every option in `Opts`, its matches appear in the order as they came in `Arguments`.

Assuming `Arguments` is `['-file', 'input.txt', '-file', 'input2.txt', '--output_file', 'out.txt', '-create-dir', '--decode', '--unsorte']`, some possible uses of `getopts/4` follow.

- Check that a simple option has been selected:

```
?- getopts(Args, ['create-dir'], M, R).
Args = ...
M = ['create-dir'],
R = ['-file','input.txt','-file','input2.txt','--output_file',
     'out.txt','--decode','--unsorte']
```

- Which argument was given to an option expecting an additional value?

```
1 ?- getopts(Args, [output_file/1], M, R).
Args = ...
M = [output_file('out.txt')],
R = ['-file','input.txt','-file','input2.txt','-create-dir',
     '--decode','--unsorte']

1 ?- getopts(Args, [output_file/1], [output_file(F)], R).
Args = ..
F = 'out.txt',
R = ['-file','input.txt','-file','input2.txt','-create-dir',
     '--decode','--unsorte']
```

- Extract options (and associated values) which can appear several times.

```
1 ?- getopts(Args, [file/1], M, R).
Args = ...
M = [file('input.txt'),file('input2.txt')],
R = ['--output_file','out.txt','-create-dir','--decode',
     '--unsorte']
```

- Was decoding selected?

```
1 ?- getopts(Args, [decode], [_], R).
Args = ...
R = ['-file','input.txt','-file','input2.txt','--output_file',
     'out.txt', '-create-dir','--unsorte']
```

- Was encoding selected?

```
1 ?- getopts(Args, [encoding], [_], R).
no
```

- Was decoding **not** selected?

```
1 ?- getopts(Args, [decode], [], R).
no
```

- Are all the options passed to the program legal options? If this is not the case, which option(s) is/are not legal?

```
1 ?- getopts(Args, [file/1, output_file/1, 'create-dir',
                    encode, decode, unsorted], _, R).
Args = ...
R = ['--unsorte'] ?
```

The complexity of `getopts/1` is currently *O(La x Lo)*, where *La* is the length of the argument list and *Lo* is the length of the option list.

– *Call and exit should be compatible with:*

`Arguments` is a list of `atom`s. (basic_props:list/2)

`Opts` is a list of `spec`s. (basic_props:list/2)

`Matched` is a list of `term`s. (basic_props:list/2)

`Rest` is a list of `term`s. (basic_props:list/2)

– *The following properties should hold at call time:*

`Arguments` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Opts` is currently a term which is not a free variable. (term_typing:nonvar/1)

cl_option/2: PREDICATE

Usage: `cl_option(Arguments,Option)`

- *Description:* Check that `Option` is an option in `Arguments`.
- *Call and exit should be compatible with:*

  `Arguments` is a list of `atoms`. (basic_props:list/2)

  `Option` is `AtomName/Arity` (getopts:spec/1)
- *The following properties should hold at call time:*

  `Arguments` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Option` is currently a term which is not a free variable. (term_typing:nonvar/1)

64.3 Documentation on internals (getopts)

spec/1: REGTYPE

Usage: `spec(Spec)`

- *Description:* `Spec` is `AtomName/Arity`

65 llists (library)

65.1 Usage and interface (`llists`)

- **Library usage:**
  `:- use_module(library(llists)).`
- **Exports:**
  - *Predicates:*
    `append/2`, `flatten/2`, `collect_singletons/2`, `transpose/2`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

65.2 Documentation on exports (`llists`)

append/2: PREDICATE

Usage:

- *Description:* Concatenates a list of lists into a list.
- *Call and exit should be compatible with:*
  `Arg1` is a list of `lists`. (basic_props:list/2)
  `Arg2` is a list. (basic_props:list/1)

flatten/2: PREDICATE

Usage:

- *Description:* Flattens out nested lists into a list.
- *Call and exit should be compatible with:*
  `Arg2` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*
  `Arg1` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*
  `Arg2` is a list. (basic_props:list/1)

collect_singletons/2: PREDICATE

Usage:

- *Description:* Collects in a list the singletons lists appearing in a list of lists.
- *Call and exit should be compatible with:*
  `Arg2` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*
  `Arg1` is a list of `lists`. (basic_props:list/2)
- *The following properties should hold upon exit:*
  `Arg2` is a list. (basic_props:list/1)

transpose/2: PREDICATE

Usage:

- *Description:* Transposes a list of lists, that is, viewing it as a matrix changes rows by columns.
- *Call and exit should be compatible with:*

  `Arg2` is a list of `lists`. (basic_props:list/2)
- *The following properties should hold at call time:*

  `Arg1` is a list of `lists`. (basic_props:list/2)
- *The following properties should hold upon exit:*

  `Arg2` is a list of `lists`. (basic_props:list/2)

66 Structured stream handling

66.1 Usage and interface (streams)

- **Library usage:**
  `:- use_module(library(streams)).`
- **Exports:**
  - *Predicates:*
    `open_null_stream/1`, `open_input/2`, `close_input/1`, `open_output/2`, `close_output/1`.

66.2 Documentation on exports (streams)

open_null_stream/1: PREDICATE

Usage: `open_null_stream(S)`

- *The following properties should hold upon exit:*
  `S` is an open stream. (streams_basic:stream/1)

open_input/2: PREDICATE

Usage: `open_input(FileName,InputStreams)`

- *The following properties should hold at call time:*
  `FileName` is a source name. (streams_basic:sourcename/1)
- *The following properties should hold upon exit:*
  streams:input_handler(InputStreams) (streams:input_handler/1)

close_input/1: PREDICATE

Usage: `close_input(InputStreams)`

- *The following properties should hold at call time:*
  streams:input_handler(InputStreams) (streams:input_handler/1)
- *The following properties should hold upon exit:*
  streams:input_handler(InputStreams) (streams:input_handler/1)

open_output/2: PREDICATE

Usage: `open_output(FileName,OutputStreams)`

- *The following properties should hold at call time:*
  `FileName` is a source name. (streams_basic:sourcename/1)
- *The following properties should hold upon exit:*
  streams:output_handler(OutputStreams) (streams:output_handler/1)

close_output/1: PREDICATE

Usage: `close_output(OutputStreams)`

- *The following properties should hold at call time:*
  streams:output_handler(OutputStreams) (streams:output_handler/1)
- *The following properties should hold upon exit:*
  streams:output_handler(OutputStreams) (streams:output_handler/1)

67 Dictionaries

Author(s): The CLIP Group.

This module provides predicates for implementing dictionaries. Such dictionaries are currently implemented as ordered binary trees of key-value pairs.

67.1 Usage and interface (dict)

- **Library usage:**
  `:- use_module(library(dict)).`
- **Exports:**
  - *Predicates:*
    `dictionary/5`, `dic_node/2`, `dic_lookup/3`, `dic_lookup/4`, `dic_get/3`, `dic_replace/4`.
  - *Regular Types:*
    `dictionary/1`, `old_or_new/1`, `non_empty_dictionary/1`.

67.2 Documentation on exports (dict)

dictionary/1: REGTYPE

Usage: `dictionary(D)`

- *Description:* `D` is a dictionary.

dictionary/5: PREDICATE

Usage: `dictionary(D,K,V,L,R)`

- *Description:* The dictionary node `D` has key `K`, value `V`, left child `L`, and right child `R`.
- *The following properties should hold upon exit:*
  `D` is a non-empty dictionary. (dict:non_empty_dictionary/1)

dic_node/2: PREDICATE

Usage: `dic_node(D,N)`

- *Description:* `N` is a sub-dictionary of `D`.
- *The following properties should hold at call time:*
  `D` is a non-empty dictionary. (dict:non_empty_dictionary/1)
- *The following properties should hold upon exit:*
  `N` is a dictionary. (dict:dictionary/1)

dic_lookup/3: PREDICATE

Usage: `dic_lookup(D,K,V)`

- *Description:* `D` contains value `V` at key `K`. If it was not already in `D` it is added.
- *The following properties should hold upon exit:*
  `D` is a non-empty dictionary. (dict:non_empty_dictionary/1)

dic_lookup/4: PREDICATE

Usage: `dic_lookup(D,K,V,O)`

- *Description:* Same as `dic_lookup(D,K,V)`. `O` indicates if it was already in `D` (`old`) or not (`new`).
- *The following properties should hold upon exit:*

  `D` is a non-empty dictionary. (dict:non_empty_dictionary/1)

  dict:old_or_new(O) (dict:old_or_new/1)

dic_get/3: PREDICATE

Usage: `dic_get(D,K,V)`

- *Description:* `D` contains value `V` at key `K`. Fails if it is not already in `D`.
- *The following properties should hold at call time:*

  `D` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `D` is a dictionary. (dict:dictionary/1)
- *The following properties should hold upon exit:*

  `D` is a non-empty dictionary. (dict:non_empty_dictionary/1)

dic_replace/4: PREDICATE

Usage: `dic_replace(D,K,V,D1)`

- *Description:* `D` and `D1` are identical except for the element at key `K`, which in `D1` contains value `V`, whatever has (or whether it is) in `D`.
- *The following properties should hold at call time:*

  `D` is a dictionary. (dict:dictionary/1)

  `D1` is a dictionary. (dict:dictionary/1)
- *The following properties should hold upon exit:*

  `D` is a dictionary. (dict:dictionary/1)

  `D1` is a dictionary. (dict:dictionary/1)

old_or_new/1: REGTYPE

A regular type, defined as follows:

```
old_or_new(old).
old_or_new(new).
```

non_empty_dictionary/1: REGTYPE

Usage: `non_empty_dictionary(D)`

- *Description:* `D` is a non-empty dictionary.

67.3 Known bugs and planned improvements (`dict`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

68 String processing

Author(s): Daniel Cabeza.

This module provides predicates for doing input/output with strings (character code lists) and for including in grammars defining strings.

68.1 Usage and interface (strings)

- **Library usage:**
  `:- use_module(library(strings)).`
- **Exports:**
  - *Predicates:*
    `get_line/2`, `get_line/1`, `write_string/2`, `write_string/1`, `whitespace/2`, `whitespace0/2`, `string/3`.
  - *Regular Types:*
    `line/1`.

68.2 Documentation on exports (strings)

get_line/2: PREDICATE

`get_line(Stream,Line)`

Reads from `Stream` a line of text and unifies `Line` with it. The end of the line can have UNIX [10] or MS-DOS [13 10] termination, which is not included in `Line`. At EOF, the term end_of_file is returned.

Usage: `get_line(S,L)`

- *The following properties should hold at call time:*
  `S` is an open stream. (streams_basic:stream/1)
- *The following properties should hold upon exit:*
  strings:line(L) (strings:line/1)

get_line/1: PREDICATE

`get_line(Line)`

Behaves like `current_input(S), get_line(S,Line)`.

Usage: `get_line(L)`

- *The following properties should hold upon exit:*
  strings:line(L) (strings:line/1)

line/1: REGTYPE

A regular type, defined as follows:

```
line(L) :-
        string(L).
line(end_of_file).
```

write_string/2: PREDICATE

`write_string(Stream,String)`

Writes `String` onto `Stream`.

Usage: `write_string(Stream,String)`

– *The following properties should hold at call time:*

`Stream` is an open stream. (streams_basic:stream/1)

`String` is a string (a list of character codes). (basic_props:string/1)

write_string/1: PREDICATE

`write_string(String)`

Behaves like `current_input(S), write_string(S, String)`.

Usage: `write_string(String)`

– *The following properties should hold at call time:*

`String` is a string (a list of character codes). (basic_props:string/1)

whitespace/2: PREDICATE

`whitespace(String,Rest)`

In a grammar rule, as `whitespace/0`, represents whitespace (a positive number of space (32), tab (9), newline (10) or return (13) characters). Thus, `Rest` is a proper suffix of `String` with one or more whitespace characters removed. An example of use would be:

```
attrs([]) --> ""
attrs([N|Ns]) -->
    whitespace,
    attr(N),
    attrs(Ns).
```

Usage: `whitespace(S1,S2)`

– *The following properties should hold at call time:*

`S1` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold upon exit:*

`S2` is a string (a list of character codes). (basic_props:string/1)

whitespace0/2: PREDICATE

`whitespace0(String,Rest)`

In a grammar rule, as `whitespace0/0`, represents possible whitespace (any number of space (32), tab (9), newline (10) or return (13) characters). Thus, `Rest` is `String` or a proper suffix of `String` with one or more whitespace characters removed. An example of use would be:

```
assignment(N,V) -->
    variable_name(N), whitespace0, "=", whitespace0, value(V).
```

Usage: `whitespace0(S1,S2)`

– *The following properties should hold at call time:*

`S1` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold upon exit:*

`S2` is a string (a list of character codes). (basic_props:string/1)

string/3: PREDICATE

`string(String,Head,Tail)`

In a grammar rule, as `string/1`, represents literally `String`. An example of use would be:

```
double(A) -->
        string(A),
        string(A).
```

Usage 1:

– *Call and exit should be compatible with:*

`String` is a string (a list of character codes). (basic_props:string/1)

`Head` is a string (a list of character codes). (basic_props:string/1)

`Tail` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold upon exit:*

`String` is a string (a list of character codes). (basic_props:string/1)

`Head` is a string (a list of character codes). (basic_props:string/1)

`Tail` is a string (a list of character codes). (basic_props:string/1)

Usage 2: `string(A,B,C)`

– *The following properties should hold at call time:*

`C` is a list. (basic_props:list/1)

– *The following properties should hold upon exit:*

`A` is a list. (basic_props:list/1)

`B` is a list. (basic_props:list/1)

69 Printing status and error messages

Author(s): The CLIP Group.

This is a very simple library for printing status and error messages to the console.

69.1 Usage and interface (messages)

- **Library usage:**
  `:- use_module(library(messages)).`
- **Exports:**
  - *Predicates:*
    `error_message/1`, `error_message/2`, `error_message/3`, `warning_message/1`, `warning_message/2`, `warning_message/3`, `note_message/1`, `note_message/2`, `note_message/3`, `simple_message/1`, `simple_message/2`, `optional_message/2`, `optional_message/3`, `debug_message/1`, `debug_message/2`, `debug_goal/2`, `debug_goal/3`, `show_message/2`, `show_message/3`, `show_message/4`.
  - *Regular Types:*
    `message_t/1`.
  - *Multifiles:*
    `issue_debug_messages/1`.
- **Other modules used:**
  - *System library modules:*
    `format`, `lists`, `write`, `filenames`, `compiler/c_itf_internal`.

69.2 Documentation on exports (messages)

error_message/1: PREDICATE

Usage: `error_message(Text)`

- *Description:* Same as `message(error,Text)`.
- *The following properties should hold at call time:*
  `Text` is a string (a list of character codes). (basic_props:string/1)

error_message/2: PREDICATE

Meta-predicate with arguments: `error_message(?,addmodule(?))`.

Usage: `error_message(Text,ArgList)`

- *Description:* Same as `message(error,Text,ArgList)`.
- *The following properties should hold at call time:*
  `Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)
  `ArgList` is a list. (basic_props:list/1)

error_message/3: PREDICATE

Meta-predicate with arguments: `error_message(?,?,addmodule(?))`.

Usage: `error_message(Lc,Text,ArgList)`

- *Description:* Same as `message(error,Lc,Text,ArgList)`.
- *The following properties should hold at call time:*

  Identifies a source line range in a file.

  ```
  location_t(loc(File,L1,L2)) :-
          atm(File),
          int(L1),
          int(L2).
  ```

  (c_itf_internal:location_t/1)

  `Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

  `ArgList` is a list. (basic_props:list/1)

warning_message/1: PREDICATE

Usage: `warning_message(Text)`

- *Description:* Same as `message(warning,Text)`.
- *The following properties should hold at call time:*

  `Text` is a string (a list of character codes). (basic_props:string/1)

warning_message/2: PREDICATE

Meta-predicate with arguments: `warning_message(?,addmodule(?))`.

Usage: `warning_message(Text,ArgList)`

- *Description:* Same as `message(warning,Text,ArgList)`.
- *The following properties should hold at call time:*

  `Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

  `ArgList` is a list. (basic_props:list/1)

warning_message/3: PREDICATE

Meta-predicate with arguments: `warning_message(?,?,addmodule(?))`.

Usage: `warning_message(Lc,Text,ArgList)`

- *Description:* Same as `message(warning,Lc,Text,ArgList)`.
- *The following properties should hold at call time:*

  Identifies a source line range in a file.

  ```
  location_t(loc(File,L1,L2)) :-
          atm(File),
          int(L1),
          int(L2).
  ```

  (c_itf_internal:location_t/1)

  `Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

  `ArgList` is a list. (basic_props:list/1)

note_message/1: PREDICATE

Usage: `note_message(Text)`

– *Description:* Same as `message(note,Text)`.

– *The following properties should hold at call time:*

`Text` is a string (a list of character codes). (basic_props:string/1)

note_message/2: PREDICATE

Meta-predicate with arguments: `note_message(?,addmodule(?))`.

Usage: `note_message(Text,ArgList)`

– *Description:* Same as `message(note,Text,ArgList)`.

– *The following properties should hold at call time:*

`Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

`ArgList` is a list. (basic_props:list/1)

note_message/3: PREDICATE

Meta-predicate with arguments: `note_message(?,?,addmodule(?))`.

Usage: `note_message(Lc,Text,ArgList)`

– *Description:* Same as `message(note,Lc,Text,ArgList)`.

– *The following properties should hold at call time:*

Identifies a source line range in a file.

```
location_t(loc(File,L1,L2)) :-
        atm(File),
        int(L1),
        int(L2).
```

(c_itf_internal:location_t/1)

`Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

`ArgList` is a list. (basic_props:list/1)

simple_message/1: PREDICATE

Usage: `simple_message(Text)`

– *Description:* The text provided in `Text` is printed.

– *The following properties should hold at call time:*

`Text` is a string (a list of character codes). (basic_props:string/1)

simple_message/2: PREDICATE

Usage: `simple_message(Text,ArgList)`

– *Description:* The text provided in `Text` is printed as a message, using the arguments in `ArgList`.

– *The following properties should hold at call time:*

`Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

`ArgList` is a list. (basic_props:list/1)

optional_message/2: PREDICATE

Usage: `optional_message(Text,Opts)`

- *Description:* The text provided in `Text` is printed as a message, but only if the atom `-v` is a member of `Opts`. These predicates are meant to be used for optional messages, which are only to be printed when *verbose* output is requested explicitly.
- *The following properties should hold at call time:*

  `Text` is a string (a list of character codes). (basic_props:string/1)

  `Opts` is a list of `atm`s. (basic_props:list/2)

optional_message/3: PREDICATE

Usage: `optional_message(Text,ArgList,Opts)`

- *Description:* The text provided in `Text` is printed as a message, using the arguments in `ArgList`, but only if the atom `-v` is a member of `Opts`. These predicates are meant to be used for optional messages, which are only to be printed when *verbose* output is requested explicitly.
- *The following properties should hold at call time:*

  `Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

  `ArgList` is a list. (basic_props:list/1)

  `Opts` is a list of `atm`s. (basic_props:list/2)

debug_message/1: PREDICATE

Usage: `debug_message(Text)`

- *Description:* The text provided in `Text` is printed as a debugging message. These messages are turned `on` by defining a fact of `issue_debug_messages/1` with the module name as argument.
- *The following properties should hold at call time:*

  `Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

debug_message/2: PREDICATE

Meta-predicate with arguments: `debug_message(?,addmodule(?))`.

Usage: `debug_message(Text,ArgList)`

- *Description:* The text provided in `Text` is printed as a debugging message, using the arguments in `ArgList` to interpret any variable-related formatting commands embedded in `Text`. These messages are turned `on` by defining a fact of `issue_debug_messages/1` which the module name as argument.
- *The following properties should hold at call time:*

  `Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

  `ArgList` is a list. (basic_props:list/1)

debug_goal/2: PREDICATE

Usage: `debug_goal(Goal,Text)`

– *Description:* `Goal` is called. The text provided in `Text` is then printed as a debugging message. The whole process (including running `Goal`) is turned `on` by defining a fact of `issue_debug_messages/1` with the module name as argument.

debug_goal/3: PREDICATE

Meta-predicate with arguments: `debug_goal(goal,?,addmodule(?))`.

Usage: `debug_goal(Goal,Text,ArgList)`

– *Description:* `Goal` is called. The text provided in `Text` is then printed as a debugging message, using the arguments in `ArgList` to interpret any variable-related formatting commands embedded in `Text`. Note that the variables in `ArgList` can be computed by `Goal`. The whole process (including running `Goal`) is turned `on` by defining a fact of `issue_debug_messages/1` with the module name as argument.

show_message/2: PREDICATE

Usage: `show_message(Type,Text)`

– *Description:* The text provided in `Text` is printed as a message of type `Type`.

– *The following properties should hold at call time:*

The types of messaes supported by the message predicate (messages:message_t/1)

`Text` is a string (a list of character codes). (basic_props:string/1)

show_message/3: PREDICATE

Meta-predicate with arguments: `show_message(?,?,addmodule(?))`.

Usage: `show_message(Type,Text,ArgList)`

– *Description:* The text provided in `Text` is printed as a message of type `Type`, using the arguments in `ArgList` to interpret any variable-related formatting commands embedded in `Text`.

– *The following properties should hold at call time:*

The types of messaes supported by the message predicate (messages:message_t/1)

`Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

`ArgList` is a list. (basic_props:list/1)

show_message/4: PREDICATE

Meta-predicate with arguments: `show_message(?,?,?,addmodule(?))`.

Usage: `show_message(Type,Lc,Text,ArgList)`

– *Description:* The text provided in `Text` is printed as a message of type `Type`, using the arguments in `ArgList` to interpret any variable-related formatting commands embedded in `Text`, and reporting error location `Lc` (file and line numbers).

– *The following properties should hold at call time:*

The types of messaes supported by the message predicate (messages:message_t/1)

Identifies a source line range in a file.

```
location_t(loc(File,L1,L2)) :-
        atm(File),
        int(L1),
        int(L2).
```

(c_itf_internal:location_t/1)

`Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

`ArgList` is a list. (basic_props:list/1)

message_t/1: REGTYPE

Usage:

– *Description:* The types of messaes supported by the message predicate

location_t/1: (UNDOC_REEXPORT)

Imported from `c_itf_internal` (see the corresponding documentation for details).

69.3 Documentation on multifiles (messages)

issue_debug_messages/1: PREDICATE

The predicate is *multifile*.

The predicate is of type *data*.

Usage: `issue_debug_messages(Module)`

– *Description:* Printing of debugging messages is enabled for module `Module`.

– *The following properties hold upon exit:*

`Module` is an atom. (basic_props:atm/1)

69.4 Known bugs and planned improvements (messages)

- Debug message switching should really be done with an expansion, for performance.

70 Accessing and redirecting the stream aliases

Author(s): Manuel Carro.

This library allows the redefinition of the files to which the special streams `user_input`, `user_output`, and `user_error` point to. On startup they point to the standard input, standard output, and standard error, in Unix style (Windows users may find that standard error stream does not work properly). Changing the file pointed to is useful for, e.g., redirecting the place to which the Prolog's standard error stream goes from within Prolog (e.g., to start a log file).

70.1 Usage and interface (`io_alias_redirection`)

- **Library usage:**
  `:- use_module(library(io_alias_redirection)).`
- **Exports:**
  - *Predicates:*
    `set_stream/3`, `get_stream/2`.

70.2 Documentation on exports (`io_alias_redirection`)

set_stream/3: PREDICATE

Usage: `set_stream(StreamAlias,NewStream,OldStream)`

- *Description:* Associate `StreamAlias` with an open stream `newStream`. Returns in `OldStream` the stream previously associated with the alias. The mode of `NewStream` must match the intended use of `StreamAlias`.
- *The following properties should hold at call time:*

  `StreamAlias` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `NewStream` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `StreamAlias` is the alias of an open stream, i.e., an atom which represents a stream at Prolog level. (streams_basic:stream_alias/1)

  `NewStream` is an open stream. (streams_basic:stream/1)

  `OldStream` is an open stream. (streams_basic:stream/1)

get_stream/2: PREDICATE

Usage: `get_stream(StreamAlias,Stream)`

- *Description:* Return in `Stream` the stream associated with `StreamAlias`.
- *The following properties should hold at call time:*

  `StreamAlias` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `StreamAlias` is the alias of an open stream, i.e., an atom which represents a stream at Prolog level. (streams_basic:stream_alias/1)

  `Stream` is an open stream. (streams_basic:stream/1)

71 Atom to term conversion

Author(s): Francisco Bueno, Daniel Cabeza, Manuel Hermenegildo.
This module implements predicates for atom or string to term conversion.

71.1 Usage and interface (atom2term)

- **Library usage:**
  `:- use_module(library(atom2term)).`
- **Exports:**
  - *Predicates:*
    `atom2term/2`, `string2term/2`, `parse_term/3`, `parse_term/4`.

71.2 Documentation on exports (atom2term)

atom2term/2: PREDICATE

General properties: `atom2term(A,T)`

- *If the following properties should hold at call time:*
  term_basic:A=a (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:T=a (term_basic:= /2)

`atom2term(A,T)`

- *If the following properties should hold at call time:*
  term_basic:A=1 (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:T=1 (term_basic:= /2)

`atom2term(A,T)`

- *If the following properties should hold at call time:*
  term_basic:A=A (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:T=A (term_basic:= /2)

`atom2term(A,T)`

- *If the following properties should hold at call time:*
  term_basic:A=f(a) (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:T=f(a) (term_basic:= /2)

`atom2term(A,T)`

- *If the following properties should hold at call time:*
  term_basic:A=f/2 (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:T=f/2 (term_basic:= /2)

Usage: `atom2term(Atom,Term)`

- *Description:* Convert an atom into a term. `Atom` is an atom, but must have term syntax. `Term` is a term resulting from parsing `Atom` char by char. The term is assumed to be ground.
- *The following properties should hold at call time:*

  `Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Term` is a free variable. (term_typing:var/1)

string2term/2: PREDICATE

General properties: `string2term(A,T)`

- *If the following properties should hold at call time:*

  term_basic:A=[97] (term_basic:= /2)

  then the following properties should hold upon exit:

  term_basic:T=a (term_basic:= /2)

`string2term(A,T)`

- *If the following properties should hold at call time:*

  term_basic:A=[49] (term_basic:= /2)

  then the following properties should hold upon exit:

  term_basic:T=1 (term_basic:= /2)

`string2term(A,T)`

- *If the following properties should hold at call time:*

  term_basic:A=[65] (term_basic:= /2)

  then the following properties should hold upon exit:

  term_basic:T=A (term_basic:= /2)

`string2term(A,T)`

- *If the following properties should hold at call time:*

  term_basic:A=[102,40,97,41] (term_basic:= /2)

  then the following properties should hold upon exit:

  term_basic:T=f(a) (term_basic:= /2)

`string2term(A,T)`

- *If the following properties should hold at call time:*

  term_basic:A=[102,47,50] (term_basic:= /2)

  then the following properties should hold upon exit:

  term_basic:T=f/2 (term_basic:= /2)

Usage: `string2term(String,Term)`

- *Description:* Same as `atom2term/2` but first argument is a string (containing a term).
- *The following properties should hold at call time:*

  `String` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Term` is a free variable. (term_typing:var/1)

parse_term/3: PREDICATE

General properties: `parse_term(A,T,R)`

– *If the following properties should hold at call time:*
term_basic:A=[102,40,97,41] (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:T=f(a) (term_basic:= /2)
term_basic:R=[] (term_basic:= /2)

`parse_term(A,T,R)`

– *If the following properties should hold at call time:*
term_basic:A=[102,40,97,41,32,102,111,111,32] (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:T=f(a) (term_basic:= /2)
term_basic:R=[32,102,111,111,32] (term_basic:= /2)

Usage: `parse_term(String,Term,Rest)`

– *Description:* `String` is parsed into `Term` up to `Rest` (which is the non-parsed rest of the list). The term is assumed to be ground.

– *The following properties should hold at call time:*
`String` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Term` is a free variable. (term_typing:var/1)

parse_term/4: PREDICATE

General properties: `parse_term(A,T,R,V)`

– *If the following properties should hold at call time:*
term_basic:A=[102,40,88,41] (term_basic:= /2)
term_basic:V=nonvars (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:T=f(X) (term_basic:= /2)
term_basic:R=[] (term_basic:= /2)

`parse_term(A,T,R,V)`

– *If the following properties should hold at call time:*
term_basic:A=[102,40,88,41] (term_basic:= /2)
term_basic:V=vars (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:T=f([88]) (term_basic:= /2)
term_basic:R=[] (term_basic:= /2)

`parse_term(A,T,R,V)`

– *If the following properties should hold at call time:*
term_basic:A=[102,40,88,41,32,102,111,111,32] (term_basic:= /2)
term_basic:V=vars (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:T=f([88]) (term_basic:= /2)
term_basic:R=[32,102,111,111,32] (term_basic:= /2)

Usage: `parse_term(String,Term,Rest,Vars)`

- *Description:* `String` is parsed into `Term` up to `Rest` (which is the non-parsed rest of the list). The term is assumed to be ground. If `Vars` is `vars` then upper case identifiers in the term are turned into variables.
- *The following properties should hold at call time:*

  `String` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Term` is a free variable. (term_typing:var/1)

  `Vars` is currently a term which is not a free variable. (term_typing:nonvar/1)

71.3 Known bugs and planned improvements (`atom2term`)

- This is just a quick hack written mainly for parsing daVinci's messages. There should be a call to the standard reader to do this!

72 ctrlcclean (library)

72.1 Usage and interface (ctrlcclean)

- **Library usage:**
  `:- use_module(library(ctrlcclean)).`
- **Exports:**
  - *Predicates:*
    `ctrlc_clean/1`, `delete_on_ctrlc/2`, `ctrlcclean/0`.
- **Other modules used:**
  - *System library modules:*
    `system`.

72.2 Documentation on exports (ctrlcclean)

ctrlc_clean/1: PREDICATE

No further documentation available for this predicate.
Meta-predicate with arguments: `ctrlc_clean(goal)`.

delete_on_ctrlc/2: PREDICATE

No further documentation available for this predicate.

ctrlcclean/0: PREDICATE

No further documentation available for this predicate.

73 errhandle (library)

73.1 Usage and interface (errhandle)

- **Library usage:**
  `:- use_module(library(errhandle)).`
- **Exports:**
  - *Predicates:*
    `error_protect/1`, `handle_error/2`.
- **Other modules used:**
  - *System library modules:*
    `system`.

73.2 Documentation on exports (errhandle)

error_protect/1: PREDICATE

No further documentation available for this predicate.
Meta-predicate with arguments: `error_protect(goal)`.

handle_error/2: PREDICATE

No further documentation available for this predicate.

74 Fast reading and writing of terms

Author(s): Daniel Cabeza, Oscar Portela Arjona.

This library provides predicates to support reading / writing of terms on a format designed to be handled on read faster than standard representation.

74.1 Usage and interface (`fastrw`)

- **Library usage:**
  `:- use_module(library(fastrw)).`
- **Exports:**
  - *Predicates:*
    `fast_read/1`, `fast_write/1`, `fast_read/2`, `fast_write/2`, `fast_write_to_string/3`.

74.2 Documentation on exports (`fastrw`)

fast_read/1: PREDICATE

`fast_read(Term)`

The next term is read from current standard input and is unified with `Term`. The syntax of the term must agree with fast_read / fast_write format. If the end of the input has been reached, `Term` is unified with the term 'end_of_file'. Further calls to `fast_read/1` will then cause an error.

fast_write/1: PREDICATE

`fast_write(Term)`

Output `Term` in a way that `fast_read/1` and `fast_read/2` will be able to read it back.

fast_read/2: PREDICATE

`fast_read(Stream,Term)`

The next term is read from `Stream` and unified with `Term`. The syntax of the term must agree with fast_read / fast_write format. If the end of the input has been reached, `Term` is unified with the term 'end_of_file'. Further calls to `fast_read/2` will then cause an error.

Usage:

- *Call and exit should be compatible with:*
  `Term` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*
  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties should hold upon exit:*
  `Term` is any term. (basic_props:term/1)

fast_write/2: PREDICATE

`fast_write(Stream,Term)`

Output `Term` to `Stream` in a way that `fast_read/1` and `fast_read/2` will be able to read it back.

Usage:

- *The following properties should hold at call time:*

  `Stream` is an open stream. (streams_basic:stream/1)

  `Term` is any term. (basic_props:term/1)

- *The following properties should hold upon exit:*

  `Term` is any term. (basic_props:term/1)

- *The following properties should hold globally:*

  `Term` is not further instantiated. (basic_props:not_further_inst/2)

fast_write_to_string/3: PREDICATE

No further documentation available for this predicate.

74.3 Known bugs and planned improvements (`fastrw`)

- Both `fast_read/2` and `fast_write/2` simply set the current output/input and call `fast_read/1` and `fast_write/1`. Therefore, in the event an error hapens during its execution, the current input / output streams may be left pointing to the `Stream`

75 File name manipulation

Author(s): Daniel Cabeza, Angel Fernandez Pineda.

This library provides some small utilities to handle file name syntax.

75.1 Usage and interface (`filenames`)

- **Library usage:**
  `:- use_module(library(filenames)).`
- **Exports:**
  - *Predicates:*
    `no_path_file_name/2`, `file_directory_base_name/3`, `file_name_extension/3`, `basename/2`, `extension/2`.
  - *Regular Types:*
    `atom_or_str/1`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`, `lists`.

75.2 Documentation on exports (`filenames`)

no_path_file_name/2: PREDICATE

This predicate will extract the last item (usually the file name) from a given path.

The first argument must be instantiated to a string or atom. Whenever the first argument is an atom, the second argument will be an atom. Whenever the first argument is a string, the second argument will be a string.

This predicate will fail under any of the following conditions:

- First argument is not an atom, nor a string.
- Second argument is not the last given path item (given path is the first argument).

Those are the most usual usages of no_path_file_name/2:

```
?- no_path_file_name("/home/nexusV/somefile.txt",K).

K = "somefile.txt" ?

yes
?- no_path_file_name('/home/nexusV/somefile.txt',K).

K = 'somefile.txt' ?

yes
?-
```

Usage: `no_path_file_name(Path,FileName)`

– *Description:* `FileName` is the file corresponding to the given `Path`.

– *Call and exit should be compatible with:*
`Path` is an atom or a string (filenames:atom_or_str/1)
`FileName` is an atom or a string (filenames:atom_or_str/1)

file_directory_base_name/3: PREDICATE

Usage: `file_directory_base_name(Path,Directory,BaseName)`

– *Description:* Given a file path `Path`, `Directory` is the directory part and `BaseName` is the filename part. `Directory` does not end in '/' unless it is just '/'. `Directory` is '.' if `Path` does not contain '/'.
– *Call and exit should be compatible with:*
`Path` is an atom or a string (filenames:atom_or_str/1)
`Directory` is an atom or a string (filenames:atom_or_str/1)
`BaseName` is an atom or a string (filenames:atom_or_str/1)

file_name_extension/3: PREDICATE

This predicate may be used in two ways:

- To create a file name from its components: name and extension. For instance:

```
?- file_name_extension(File,mywork,'.txt').

File = 'mywork.txt' ?

yes
?-
```

- To split a file name into its name and extension. For Instance:

```
?- file_name_extension('mywork.txt',A,B).

A = mywork,
B = '.txt' ?

yes
?-
```

Any other usage of file_name_extension/3 will cause the predicate to fail. Notice that valid arguments are accepted both as atoms or strings.

General properties: `file_name_extension(File,Name,Ext)`

– *Description:* This is a bug, this test must succeeds.
– *If the following properties do not hold at call time:*
term_basic:File=/home/user/emacs.d/dummy (term_basic:= /2)
then the following properties do not hold upon exit:
term_basic:Name=/home/user/emacs.d/dummy (term_basic:= /2)
term_basic:Ext= (term_basic:= /2)
then the following properties do not hold globally:
All calls of the form `file_name_extension(File,Name,Ext)` are deterministic. (native_props:is_det/1)
All the calls of the form `file_name_extension(File,Name,Ext)` do not fail. (native_props:not_fails/1)

Usage: `file_name_extension(FileName,BaseName,Extension)`

– *Description:* Splits a `FileName` into its `BaseName` and `Extension`.

– *Call and exit should be compatible with:*

`FileName` is an atom or a string	(filenames:atom_or_str/1)
`BaseName` is an atom or a string	(filenames:atom_or_str/1)
`Extension` is an atom or a string	(filenames:atom_or_str/1)

basename/2: PREDICATE

`basename(FileName,BaseName)`

`BaseName` is `FileName` without extension. Equivalent to `file_name_extension(FileName,BaseName,_)`. Useful to extract the base name of a file using functional syntax.

Usage:

– *Call and exit should be compatible with:*

`FileName` is an atom or a string	(filenames:atom_or_str/1)
`BaseName` is an atom or a string	(filenames:atom_or_str/1)

atom_or_str/1: REGTYPE

Usage: `atom_or_str(X)`

– *Description:* `X` is an atom or a string

extension/2: PREDICATE

`extension(FileName,Extension)`

`Extension` is the extension (suffix) of `FileName`. Equivalent to `file_name_extension(FileName,_,Extension)`. Useful to extract the extension of a file using functional syntax.

Usage:

– *Call and exit should be compatible with:*

`FileName` is an atom or a string	(filenames:atom_or_str/1)
`Extension` is an atom or a string	(filenames:atom_or_str/1)

76 Symbolic filenames

Author(s): Francisco Bueno.

This module provides a predicate for file opening which can use any term as an alias for the filename (i.e., symbolic filenames) instead of the usual constants which are file system path names of the actual files.

The correspondence between an alias and the actual file path is done dynamically, without having to recompile the program. It is possible to define the correspondence via facts for `file_alias/2` in a file declared with `multifile:alias_file/1` in the program: those facts will be dynamically loaded when running the program. Alternatively, the correspondence can be defined via shell environment variables, by defining the value of a variable by the (symbolic) name of the file to be the path of the actual file.

76.1 Usage and interface (`symfnames`)

- **Library usage:**
  `:- use_module(library(symfnames)).`
- **Exports:**
  - *Predicates:*
    `open/3`.
  - *Multifiles:*
    `alias_file/1`, `file_alias/2`.
- **Other modules used:**
  - *System library modules:*
    `read`, `system`.

76.2 Documentation on exports (`symfnames`)

open/3: PREDICATE

`open(File,Mode,Stream)`

Open `File` with mode `Mode` and return in `Stream` the stream associated with the file. It is like `streams_basic:open/3`, but `File` is considered a symbolic name: either defined by `user:file_alias/2` or as an environment variable. Predicate `user:file_alias/2` is inspected before the environment variables.

Usage:

- *Calls should, and exit will be compatible with:*
  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties should hold at call time:*
  `File` is any term. (basic_props:term/1)
  `Mode` is an opening mode ('read', 'write' or 'append'). (streams_basic:io_mode/1)
- *The following properties hold upon exit:*
  `Stream` is an open stream. (streams_basic:stream/1)

76.3 Documentation on multifiles (symfnames)

alias_file/1: PREDICATE

`alias_file(File)`

Declares `File` to be a file defining symbolic names via `file_alias/2`. Anything else in `File` is simply ignored.

The predicate is *multifile*.

file_alias/2: PREDICATE

`file_alias(Alias,File)`

Declares `Alias` as a symbolic name for `File`, the real name of an actual file (or directory).

The predicate is *multifile*.

The predicate is of type *data*.

76.4 Other information (symfnames)

The example discussed here is included in the distribution files. There is a main application file which uses module `mm`. This module reads a line from a file; the main predicate in the main file then prints this line. The important thing is that the file read is named by a symbolic name `"file"`. The main application file declares another file where the symbolic names are assigned actual file names:

```
:- use_module(mm).

:- multifile alias_file/1.
alias_file(myfiles).

main :- p(X), display(X), nl.
```

Now, the file `myfiles.pl` can be used to change the file you want to read from without having to recompile the application. The current assignment is:

```
%:- use_package([]).
file_alias(file,'mm.pl').
```

so the execution of the application will show the first line of `mm.pl`. However, you can change to:

```
file_alias(file,'main.pl').
```

and then execution of the same executable will show the first line of `main.pl`.

77 File I/O utilities

Author(s): The CLIP Group.
This module implements the file I/O utilities.

77.1 Usage and interface (`file_utils`)

- **Library usage:**
  `:- use_module(library(file_utils)).`
- **Exports:**
  - *Predicates:*
    `file_terms/2`, `copy_stdout/1`, `file_to_string/2`, `file_to_string/3`, `string_to_file/2`, `stream_to_string/2`, `stream_to_string/3`, `output_to_file/2`.
- **Other modules used:**
  - *System library modules:*
    `read`, `streams`, `strings`.

77.2 Documentation on exports (`file_utils`)

file_terms/2: PREDICATE

Usage 1: `file_terms(File,Terms)`

- *Description:* Unifies `Terms` with the list of all terms in `File`.
- *The following properties should hold at call time:*
  `File` is a source name. (streams_basic:sourcename/1)
  `Terms` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  `Terms` is a list. (basic_props:list/1)

Usage 2: `file_terms(File,Terms)`

- *Description:* Writes the terms in list `Terms` (including the ending '.') onto file `File`.
- *The following properties should hold at call time:*
  `File` is a source name. (streams_basic:sourcename/1)
  `Terms` is a list. (basic_props:list/1)

copy_stdout/1: PREDICATE

Usage: `copy_stdout(File)`

- *Description:* Copies file `File` to standard output.
- *The following properties should hold at call time:*
  `File` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `File` is a source name. (streams_basic:sourcename/1)

file_to_string/2: PREDICATE

Usage: `file_to_string(FileName,String)`

- *Description:* Reads all the characters from the file `FileName` and returns them in `String`.
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `String` is a free variable. (term_typing:var/1)

  `FileName` is a source name. (streams_basic:sourcename/1)
- *The following properties should hold upon exit:*

  `String` is a string (a list of character codes). (basic_props:string/1)

file_to_string/3: PREDICATE

Usage: `file_to_string(FileName,String,Tail)`

- *Description:* Reads all the characters from the file `FileName` and returns them in `String`. `Tail` is the end of `String`.
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `String` is a free variable. (term_typing:var/1)

  `FileName` is a source name. (streams_basic:sourcename/1)
- *The following properties should hold upon exit:*

  `String` is a string (a list of character codes). (basic_props:string/1)

string_to_file/2: PREDICATE

Usage: `string_to_file(String,FileName)`

- *Description:* Reads all the characters from the string `String` and writes them to file `FileName`.
- *The following properties should hold at call time:*

  `String` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `String` is a string (a list of character codes). (basic_props:string/1)

  `FileName` is a source name. (streams_basic:sourcename/1)

stream_to_string/2: PREDICATE

Usage: `stream_to_string(Stream,String)`

- *Description:* Reads all the characters from `Stream`, returns them in `String`, and closes `Stream`.
- *The following properties should hold at call time:*

  `Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `String` is a free variable. (term_typing:var/1)

  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties should hold upon exit:*

  `String` is a string (a list of character codes). (basic_props:string/1)

stream_to_string/3: PREDICATE

Usage: `stream_to_string(Stream,String,Tail)`

– *Description:* Reads all the characters from `Stream`, returns them in `String`, and closes `Stream`. `Tail` is the end of `String`

– *The following properties should hold at call time:*

`Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)

`String` is a free variable. (term_typing:var/1)

`Stream` is an open stream. (streams_basic:stream/1)

output_to_file/2: PREDICATE

No further documentation available for this predicate.

Meta-predicate with arguments: `output_to_file(goal,?)`.

78 File locks

Author(s): José Manuel Gómez Pérez, Daniel Cabeza, Manuel Carro.

This module implements file locks: the ability to lock a fiel so that other processes cannot access it until the file is unlocked. **It is, however, not working.** The predicates do nothing. Proper implementation is planned for a near future.

78.1 Usage and interface (`file_locks`)

- **Library usage:**
  `:- use_module(library(file_locks)).`
- **Exports:**
  - *Predicates:*
    `lock_file/3`, `unlock_file/2`.

78.2 Documentation on exports (`file_locks`)

lock_file/3: PREDICATE

Usage: `lock_file(File,LockType,Result)`

- *Description:* Tries to lock `File` with `LockType` and returns the result (either `true` or `false`) in `Result`.
- *Call and exit should be compatible with:*
  `File` is an atom. (basic_props:atm/1)
  `LockType` is an atom. (basic_props:atm/1)
  `Result` is an atom. (basic_props:atm/1)

unlock_file/2: PREDICATE

Usage: `unlock_file(File,Result)`

- *Description:* Tries to unlock `File` the result (either `true` or `false`) in `Result`.
- *Call and exit should be compatible with:*
  `File` is an atom. (basic_props:atm/1)
  `Result` is an atom. (basic_props:atm/1)

78.3 Known bugs and planned improvements (`file_locks`)

- No doing anything helpful.

79 Lists and conjunctions and disjunctions

79.1 Usage and interface (formulae)

- **Library usage:**
  `:- use_module(library(formulae)).`
- **Exports:**
  - *Predicates:*
    `list_to_conj/3`, `list_to_conj/2`, `conj_to_list/2`, `list_to_disj/2`, `disj_to_list/2`, `conj_to_llist/2`, `llist_to_conj/2`, `disj_to_llist/2`, `llist_to_disj/2`, `body2list/2`, `asbody_to_conj/2`, `list_to_disj2/2`.
  - *Properties:*
    `assert_body_type/1`.
  - *Regular Types:*
    `conj_disj_type/1`, `t_conj/1`, `t_disj/1`.
- **Other modules used:**
  - *System library modules:*
    `messages`.

79.2 Documentation on exports (formulae)

list_to_conj/3: PREDICATE

`list_to_conj(List,Conj,End)`

`Conj` is the conjunction made up of the elements of `List` plus a final element `End`.

list_to_conj/2: PREDICATE

Usage 1: `list_to_conj(A,B)`

- *Description:* `Conj` is the conjunction made up of the elements of `List`. (`[]` is `true`). It runs in both ways.

```
?- list_to_conj( A , a ).

A = [a] ? ;

no
?- list_to_conj( A , (a,V) ).

A = [a,V] ? ;

no
?- list_to_conj( A , (a,V,b) ).

A = [a,V,b] ? ;
```

```
no
?- list_to_conj( [A] , B ).

B = A ? ;

no
?- list_to_conj( [a,A] , B ).

B = (a,A) ? ;

no
?- list_to_conj( [a,A,b] , B ).

B = (a,A,b) ? ;

no
?- list_to_conj( [] , B ).

B = true ? ;

no
```

– *The following properties should hold at call time:*
`A` is a list. (basic_props:list/1)
`B` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
Conjuntions. (formulae:t_conj/1)

Usage 2: `list_to_conj(A,B)`
– *The following properties should hold at call time:*
`A` is a free variable. (term_typing:var/1)
Conjuntions. (formulae:t_conj/1)
– *The following properties should hold upon exit:*
`A` is a list. (basic_props:list/1)

conj_to_list/2: PREDICATE

`conj_to_list(Conj,List)`

`List` is the list made up of the elements of conjunction `Conj` (`true` is `[]`).

list_to_disj/2: PREDICATE

Usage: `list_to_disj(A,B)`

– *Description:* `Disj` is the disjunction made up of the elements of `List`. (`[]` is `true`). It runs in both ways. Examples:

```
?- list_to_disj( [a] , A ).

A = a ? ;

no
```

```
?- list_to_disj( [a,b] , A ).

A = (a;b) ? ;

no
?- list_to_disj( [a,B,b] , A ).

A = (a;B;b) ? ;

no
?- list_to_disj( [a,b,B] , A ).

A = (a;b;B) ? ;

no
?- list_to_disj( A , (a) ).

A = [a] ? ;

no
?- list_to_disj( A , (a;b) ).

A = [a,b] ? ;

no
?- list_to_disj( A , (a;B;b) ).

A = [a,B,b] ? ;

no
?-
```

– *The following properties should hold at call time:*
`A` is a list. (basic_props:list/1)
`B` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
Conjuntions. (formulae:t_disj/1)

disj_to_list/2: PREDICATE

`disj_to_list(Disj,List)`

`List` is the list made up of the elements of disjunction `Disj` (`true` is `[]`).

conj_to_llist/2: PREDICATE

Turns a conjunctive (normal form) formula into a list (of lists of ...). As a side-effect, inner conjunctions get flattened. No special care for `true`.

llist_to_conj/2: PREDICATE

Inverse of `conj_to_llist/2`. No provisions for anything else than a non-empty list on input (i.e., they will go 'as are' in the output.

disj_to_llist/2: PREDICATE

Turns a disjunctive (normal form) formula into a list (of lists of ...). As a side-effect, inner disjunctions get flattened. No special care for `true`.

llist_to_disj/2: PREDICATE

Inverse of `disj_to_llist/2`. No provisions for anything else than a non-empty list on input (i.e., they will go 'as are' in the output.

body2list/2: PREDICATE

No further documentation available for this predicate.

asbody_to_conj/2: PREDICATE

Usage 1: `asbody_to_conj(A,B)`

- *Description:* Transforms assertion body `A` into a conjuntion (`B`). It runs in both ways
- *The following properties should hold at call time:*

  formulae:assert_body_type(A) (formulae:assert_body_type/1)

  `B` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  The usual prolog way of writting conjuntions and disjuntions in a body using ',' and ';' (formulae:conj_disj_type/1)

Usage 2: `asbody_to_conj(A,B)`

- *The following properties should hold at call time:*

  `A` is a free variable. (term_typing:var/1)

  The usual prolog way of writting conjuntions and disjuntions in a body using ',' and ';' (formulae:conj_disj_type/1)
- *The following properties should hold upon exit:*

  formulae:assert_body_type(A) (formulae:assert_body_type/1)

assert_body_type/1: PROPERTY

A property, defined as follows:

```
assert_body_type(X) :-
        list(X,assert_body_type__).
```

conj_disj_type/1: REGTYPE

Usage:

- *Description:* The usual prolog way of writting conjuntions and disjuntions in a body using ',' and ';'

t_conj/1: REGTYPE

Usage:

- *Description:* Conjuntions.

t_disj/1: REGTYPE

Usage:

– *Description:* Conjuntions.

list_to_disj2/2: PREDICATE

No further documentation available for this predicate.

80 Term manipulation utilities

Author(s): The CLIP Group.
This module implements some utils to do term manipulation.

80.1 Usage and interface (terms)

- **Library usage:**
  `:- use_module(library(terms)).`
- **Exports:**
  - *Predicates:*
    `term_size/2`, `copy_args/3`, `arg/2`, `atom_concat/2`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

80.2 Documentation on exports (terms)

term_size/2: PREDICATE

General properties: `term_size(A,B)`

- *If the following properties should hold at call time:*
  term_basic:A=p(a,b,c(d,e)) (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:B=6 (term_basic:= /2)

`term_size(A,B)`

- *If the following properties should hold at call time:*
  term_basic:A=p(a,b,c(d,_120328)) (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:B=6 (term_basic:= /2)

`term_size(A,B)`

- *If the following properties should hold at call time:*
  term_basic:A=[1,2,3] (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:B=7 (term_basic:= /2)

Usage: `term_size(Term,N)`

- *Description:* Determines the size of a term.
- *The following properties should hold at call time:*
  `Term` is any term. (basic_props:term/1)
  `N` is a non-negative integer. (basic_props:nnegint/1)

copy_args/3: PREDICATE

General properties: `copy_args(N,Term,Copy)`

– *If the following properties hold at call time:*
`N` is currently ground (it contains no variables). (term_typing:ground/1)
`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)
then the following properties hold globally:
`copy_args(N,Term,Copy)` is evaluable at compile-time. (basic_props:eval/1)

`copy_args(N,Term,Copy)`

– *The following properties hold globally:*
`copy_args(N,Term,Copy)` is side-effect `free`. (basic_props:sideff/2)

Usage: `copy_args(N,Term,Copy)`

– *Description:* `Term` and `Copy` have the same first `N` arguments.
– *The following properties should hold at call time:*
`N` is a non-negative integer. (basic_props:nnegint/1)

arg/2: PREDICATE

Usage: `arg(Term,Arg)`

– *Description:* `Arg` is an argument of `Term`. Gives each of the arguments on backtracking.

atom_concat/2: PREDICATE

`atom_concat(Atms,Atm)`

`Atm` is the atom resulting from concatenating all atoms in the list `Atms` in the order in which they appear. If `Atm` is an atom at call then `Atms` can contain free variables.

General properties: `atom_concat(A,B)`

– *If the following properties should hold at call time:*
term_basic:A=[/home/edison/svn/dist/CiaoDE/bin,/,X,-,1.11,] (term_basic:= /2)
term_basic:B=/home/edison/svn/dist/CiaoDE/bin/fileinfo-1.11 (term_basic:= /2)
then the following properties should hold upon exit:
The terms `X` and `fileinfo` are strictly identical. (term_compare:== /2)

`atom_concat(A,B)`

– *If the following properties should hold at call time:*
term_basic:A=[a,b,c] (term_basic:= /2)
then the following properties should hold upon exit:
The terms `B` and `abc` are strictly identical. (term_compare:== /2)

`atom_concat(X,Y)`

– *Description:* atom_concat that generates several solutions.
– *If the following properties should hold at call time:*
term_basic:X=[a,B|C] (term_basic:= /2)
term_basic:Y=abcde (term_basic:= /2)
then the following properties should hold upon exit:
`B,C` is an element of `[(,[b,c,d,e]),(,[b,c,de]),(,[b,cd,e]),(,[b,cde]),(,[bc,d,e]),(,[bc,de]),(,[bcd,e]`
(basic_props:member/2)

`atom_concat(X,Y)`

- *Description:* atom_concat that generates several solutions.
- *If the following properties should hold at call time:*
  term_basic:X=[a,B|C] (term_basic:= /2)
  term_basic:Y=abcde (term_basic:= /2)
  then the following properties should hold globally:
  Goal `atom_concat(X,Y)` produces the solutions listed in `[atom_concat([a,,b,c,d,e],abcde),atom_concat([a,,b,c,de],abcde),atom_concat([a,,b,cd,e],abcde),atom_concat([a,,b,cde],abcde),atom_concat([a,,bc,d,e],abcde),atom_concat([a,,bc,de],abcde),atom_concat([a,,bcd,e],abcde),atom_concat([a,,bcde],abcde),atom_concat([a,b,c,d,e],abcde),atom_concat([a,b,c,de],abcde),atom_concat([a,b,cd,e],abcde),atom_concat([a,b,cde],abcde),atom_concat([a,bc,d,e],abcde),atom_concat([a,bc,de],abcde),atom_concat([a,bcd,e],abcde),atom_concat([a,bcde],abcde)]`. (native_props:solutions/2)

`atom_concat(X,Y)`

- *If the following properties should hold at call time:*
  term_basic:X=[A,b,C] (term_basic:= /2)
  term_basic:Y=abc (term_basic:= /2)
  then the following properties should hold upon exit:
  The terms `A` and `a` are strictly identical. (term_compare:== /2)
  The terms `C` and `c` are strictly identical. (term_compare:== /2)

`atom_concat(X,Y)`

- *If the following properties should hold at call time:*
  term_basic:X=[A,b,] (term_basic:= /2)
  term_basic:Y=ab (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:A=a (term_basic:= /2)

Usage 1: `atom_concat(Atms,Atm)`

- *The following properties should hold at call time:*
  `Atms` is a list of `atm`s. (basic_props:list/2)
- *The following properties should hold upon exit:*
  `Atm` is an atom. (basic_props:atm/1)

Usage 2: `atom_concat(Atms,Atm)`

- *Call and exit should be compatible with:*
  `Atms` is a list of `atm`s. (basic_props:list/2)
- *The following properties should hold at call time:*
  `Atm` is an atom. (basic_props:atm/1)
- *The following properties should hold upon exit:*
  `Atms` is a list of `atm`s. (basic_props:list/2)

81 Term checking utilities

Author(s): The CLIP Group.
This module implements the term checking utilities.

81.1 Usage and interface (terms_check)

- **Library usage:**
  `:- use_module(library(terms_check)).`
- **Exports:**
  - *Predicates:*
    `ask/2`, `variant/2`, `most_general_instance/3`, `most_specific_generalization/3`.
  - *Properties:*
    `instance/2`.

81.2 Documentation on exports (terms_check)

ask/2: PREDICATE

`ask(Term1,Term2)`

`Term1` and `Term2` unify without producing bindings for the variables of `Term1`. I.e., `instance(Term1,Term2)` holds.

instance/2: PROPERTY

Usage: `instance(Term1,Term2)`

- *Description:* `Term1` is an instance of `Term2`.
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

variant/2: PREDICATE

`variant(Term1,Term2)`

`Term1` and `Term2` are identical up to renaming.

most_general_instance/3: PREDICATE

`most_general_instance(Term1,Term2,Term)`

`Term` satisfies `instance(Term,Term1)` and `instance(Term,Term2)` and there is no term more general than `Term` (modulo variants) that satisfies it.

most_specific_generalization/3: PREDICATE

`most_specific_generalization(Term1,Term2,Term)`

`Term` satisfies `instance(Term1,Term)` and `instance(Term2,Term)` and there is no term less general than `Term` (modulo variants) that satisfies it.

81.3 Other information (terms_check)

Currently, `ask/2` and `instance/2` are exactly the same. However, `ask/2` is more general, since it is also applicable to constraint domains (although not yet implemented): for the particular case of Herbrand terms, it is just `instance/2` (which is the only ask check currently implemented).

81.4 Known bugs and planned improvements (terms_check)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

82 Sets of variables in terms

Author(s): The CLIP Group.

This module implements predicates to handle sets of variables in terms.

82.1 Usage and interface (terms_vars)

- **Library usage:**
  `:- use_module(library(terms_vars)).`
- **Exports:**
  - *Predicates:*
    `varset/2`, `intersect_vars/3`, `member_var/2`, `diff_vars/3`, `varsbag/3`, `varset_in_args/2`.
- **Other modules used:**
  - *System library modules:*
    `idlists`, `sort`.

82.2 Documentation on exports (terms_vars)

varset/2: PREDICATE

`varset(Term,Xs)`

`Xs` is the sorted list of all the variables in `Term`.

intersect_vars/3: PREDICATE

No further documentation available for this predicate.

member_var/2: PREDICATE

No further documentation available for this predicate.

diff_vars/3: PREDICATE

No further documentation available for this predicate.

varsbag/3: PREDICATE

`varsbag(Term,Vs,Xs)`

`Vs` is the list of all the variables in `Term` ordered as they appear in `Term` right-to-left depth-first (including duplicates) plus `Xs`.

varset_in_args/2: PREDICATE

Usage: `varset_in_args(T,LL)`

- *Description:* Each list of `LL` contains the variables of an argument of `T`, for each argument, and in left to right order.
- *The following properties should hold at call time:*

  `T` is currently a term which is not a free variable. (term_typing:nonvar/1)
- *The following properties should hold upon exit:*

  `LL` is a list of `list(var)`s. (basic_props:list/2)

83 Cyclic terms handling

Author(s): Daniel Cabeza.

This module implements predicates related to cyclic terms. Cyclic (or infinite) terms are produced when unifying a variable with a term which contains that variable.

83.1 Usage and interface (`cyclic_terms`)

- **Library usage:**
  `:- use_module(library(cyclic_terms)).`
- **Exports:**
  - *Predicates:*
    `cyclic_term/1`, `acyclic_term/1`, `uncycle_term/2`, `recycle_term/2`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

83.2 Documentation on exports (`cyclic_terms`)

cyclic_term/1: PREDICATE

Usage: `cyclic_term(T)`

- *Description:* True if `T` is cyclic (infinite).

acyclic_term/1: PREDICATE

Usage: `acyclic_term(T)`

- *Description:* True if `T` is acyclic (finite).

uncycle_term/2: PREDICATE

Usage: `uncycle_term(T,U)`

- *Description:* Given a term `T`, `U` is a finite representation of `T` as an acyclic term. This representation can be converted back to `T` using `recycle_term/2`.

recycle_term/2: PREDICATE

Usage: `recycle_term(U,T)`

- *Description:* Given `U`, a finite representation of a term as an acyclic term as `uncycle_term/2` produces, `T` is the represented term. `U` is modified by the predicate, thus to maintain it untouched `copy_term/2` should be used.

84 A simple pretty-printer for Ciao programs

Author(s): The CLIP Group.

This library module writes out to standard output a clause or a list of clauses.

84.1 Usage and interface (pretty_print)

- **Library usage:**
  `:- use_module(library(pretty_print)).`
- **Exports:**
  - *Predicates:*
    `pretty_print/2`, `pretty_print/3`, `pretty_print/4`.
- **Other modules used:**
  - *System library modules:*
    `operators`, `vndict`, `write`.

84.2 Documentation on exports (pretty_print)

pretty_print/2: PREDICATE

Usage: `pretty_print(Cls,Flags)`

- *Description:* Prints each clause in the list `Cls` after numbering its variables.
- *The following properties should hold at call time:*
  pretty_print:clauses(Cls) (pretty_print:clauses/1)
  `Flags` is a list of `flag`s. (basic_props:list/2)

pretty_print/3: PREDICATE

Usage: `pretty_print(Cls,Flags,Ds)`

- *Description:* Prints each clause in the list `Cls` after using the corresponding variable names dictionary in `Ds` to name its variables.
- *The following properties should hold at call time:*
  pretty_print:clauses(Cls) (pretty_print:clauses/1)
  `Flags` is a list of `flag`s. (basic_props:list/2)
  `Ds` is a dictionary of variable names. (vndict:varnamedict/1)

pretty_print/4: PREDICATE

No further documentation available for this predicate.

84.3 Documentation on internals (pretty_print)

clauses/1: REGTYPE

A regular type, defined as follows:

```
clauses([]).
clauses([_1|_2]) :-
        clause(_1),
        clauses(_2).
clauses(_1) :-
        clause(_1).
```

clause/1: REGTYPE

A regular type, defined as follows:

```
clause(_1) :-
        clterm(_1).
clause((_1,_2)) :-
        clterm(_1),
        term(_2).
```

clterm/1: REGTYPE

A regular type, defined as follows:

```
clterm(clause(_1,_2)) :-
        callable(_1),
        body(_2).
clterm(directive(_1)) :-
        body(_1).
clterm((_1:-_2)) :-
        callable(_1),
        body(_2).
clterm(_1) :-
        callable(_1).
```

body/1: REGTYPE

A well formed body, including cge expressions and &-concurrent expressions. The atomic goals may or may not have a key in the form `^(goal:any)`, and may or may not be module qualified, but if they are it has to be in the form `^(^(moddesc:goal):any)`.

Usage: `body(X)`

– *Description:* `X` is a printable body.

flag/1: REGTYPE

A keyword `ask/1` flags whether to output *asks* or *whens* and `nl/1` whether to separate clauses with a blank line or not.

Usage: `flag(X)`

– *Description:* `X` is a flag for the pretty-printer.

84.4 Known bugs and planned improvements (pretty_print)

- 2.- If the priority of and operator, &/1 or &/2, is redefined with lower priority than :-/2 or ./1, the written term is incorrect because it does not include parenthesis to make Ciao associate and operator first.

85 Pretty-printing assertions

Author(s): Francisco Bueno.

This module defines some predicates which are useful for writing assertions in a readable form.

85.1 Usage and interface (assrt_write)

- **Library usage:**
  `:- use_module(library(assrt_write)).`
- **Exports:**
  - *Predicates:*
    `write_assertion/6`, `write_assertion/7`, `write_assertion_as_comment/6`, `write_assertion_as_comment/7`, `write_assertion_as_double_comment/6`, `write_assertion_as_double_comment/7`.
- **Other modules used:**
  - *System library modules:*
    `format`, `assertions/assrt_lib`, `messages`, `assertions/assertions_props`, `vndict`.

85.2 Documentation on exports (assrt_write)

write_assertion/6: PREDICATE

Usage: `write_assertion(Goal,Status,Type,Body,Dict,Flag)`

- *Description:* Writes the (normalized) assertion to current output.
- *Call and exit should be compatible with:*

`Status` is an acceptable status for an assertion.	(assertions_props:assrt_status/1)
`Type` is an admissible kind of assertion.	(assertions_props:assrt_type/1)
`Body` is a normalized assertion body.	(assertions_props:nabody/1)
`Dict` is a dictionary.	(dict:dictionary/1)
`Flag` is `status` or `nostatus`.	(assrt_write:status_flag/1)

write_assertion/7: PREDICATE

Usage: `write_assertion(Stream,Goal,Status,Type,Body,Dict,Flag)`

- *Description:* Writes the (normalized) assertion to stream `Stream`.
- *Call and exit should be compatible with:*

`Status` is an acceptable status for an assertion.	(assertions_props:assrt_status/1)
`Type` is an admissible kind of assertion.	(assertions_props:assrt_type/1)
`Body` is a normalized assertion body.	(assertions_props:nabody/1)
`Dict` is a dictionary.	(dict:dictionary/1)
`Flag` is `status` or `nostatus`.	(assrt_write:status_flag/1)

write_assertion_as_comment/6: PREDICATE

Usage: `write_assertion_as_comment(Goal,Status,Type,Body,Dict,Flag)`

- *Description:* Writes the (normalized) assertion to current output as a Prolog comment.
- *Call and exit should be compatible with:*

`Status` is an acceptable status for an assertion.	(assertions_props:assrt_status/1)
`Type` is an admissible kind of assertion.	(assertions_props:assrt_type/1)
`Body` is a normalized assertion body.	(assertions_props:nabody/1)
`Dict` is a dictionary.	(dict:dictionary/1)
`Flag` is `status` or `nostatus`.	(assrt_write:status_flag/1)

write_assertion_as_comment/7: PREDICATE

Usage: `write_assertion_as_comment(Stream,Goal,Status,Type,Body,Dict,Flag)`

- *Description:* Writes the (normalized) assertion to stream `Stream` as a Prolog comment.
- *Call and exit should be compatible with:*

`Status` is an acceptable status for an assertion.	(assertions_props:assrt_status/1)
`Type` is an admissible kind of assertion.	(assertions_props:assrt_type/1)
`Body` is a normalized assertion body.	(assertions_props:nabody/1)
`Dict` is a dictionary.	(dict:dictionary/1)
`Flag` is `status` or `nostatus`.	(assrt_write:status_flag/1)

write_assertion_as_double_comment/6: PREDICATE

Usage: `write_assertion_as_double_comment(Goal,Status,Type,Body,Dict,Flag)`

- *Description:* Writes the (normalized) assertion to current output as a Prolog double comment.
- *Call and exit should be compatible with:*

`Status` is an acceptable status for an assertion.	(assertions_props:assrt_status/1)
`Type` is an admissible kind of assertion.	(assertions_props:assrt_type/1)
`Body` is a normalized assertion body.	(assertions_props:nabody/1)
`Dict` is a dictionary.	(dict:dictionary/1)
`Flag` is `status` or `nostatus`.	(assrt_write:status_flag/1)

write_assertion_as_double_comment/7: PREDICATE

Usage:
`write_assertion_as_double_comment(Stream,Goal,Status,Type,Body,Dict,Flag)`

- *Description:* Writes the (normalized) assertion to stream `Stream` as a Prolog double comment.
- *Call and exit should be compatible with:*

`Status` is an acceptable status for an assertion.	(assertions_props:assrt_status/1)
`Type` is an admissible kind of assertion.	(assertions_props:assrt_type/1)
`Body` is a normalized assertion body.	(assertions_props:nabody/1)
`Dict` is a dictionary.	(dict:dictionary/1)
`Flag` is `status` or `nostatus`.	(assrt_write:status_flag/1)

86 The Ciao library browser

Author(s): Angel Fernandez Pineda.

The `librowser` library provides a set of predicates wich enable the user to interactively find Ciao libraries and/or any predicate exported by them.

This is a simple example:

```
?- apropos('*find*').
persdbrt_sql: dbfindall/4
persdbrtsql: dbfindall/4
conc_aggregates: findall/3
linda: rd_findall/3
vndict: find_name/4
internals: $find_file/8
aggregates: findall/4,findall/3

yes
?-
```

Librowser is specially useful when inside GNU Emacs: just place the cursor over a librowser response and press C-cTAB in order to get help on the related predicate. Refer to the **"Using Ciao inside GNU Emacs"** chapter for further information.

86.1 Usage and interface (`librowser`)

- **Library usage:**

  It is not necesary to use this library at user programs. It is designed to be used at the Ciao *toplevel* shell: `ciaosh`. In order to do so, just make use of `use_module/1` as follows:

  `use_module(library(librowser)).`

  Then, the library interface must be read. This is automatically done when calling any predicate at librowser, and the entire process will take a little moment.So, you should want to perform such a process after loading the Ciao toplevel:

  ```
  Ciao 0.9 #75: Fri Apr 30 19:04:24 MEST 1999
  ?- use_module(library(librowser)).

  yes
  ?- update.
  ```

  Whether you want this process to be automatically performed when loading `ciaosh`, you may include those lines in your *.ciaorc* personal initialization file.
- **Exports:**
  - *Predicates:*

    `update/0`, `browse/2`, `where/1`, `describe/1`, `system_lib/1`, `apropos/1`.
- **Other modules used:**
  - *System library modules:*

    `regexp/regexp_code`, `read`, `fastrw`, `system`, `streams`, `lists`.

86.2 Documentation on exports (`librowser`)

update/0: PREDICATE

This predicate will scan the Ciao system libraries for predicate definitions. This may be done once time before calling any other predicate at this library.

update/0 will also be automatically called (once) when calling any other predicate at librowser.

Usage:

– *Description:* Creates an internal database of modules at Ciao system libraries.

browse/2: PREDICATE

This predicate is fully reversible, and is provided to inspect concrete predicate specifications. For example:

```
?- browse(M,findall/A).

A = 3,
M = conc_aggregates ? ;

A = 4,
M = aggregates ? ;

A = 3,
M = aggregates ? ;

no
?-
```

Usage: `browse(Module,Spec)`

– *Description:* Asocciates the given `Spec` predicate specification with the `Module` which exports it.

– *The following properties should hold at call time:*

`Module` is a module name (an atom) (librowser:module_name/1)

`Spec` is a **Functor/Arity** predicate specification (librowser:pred_spec/1)

where/1: PREDICATE

This predicate will print at the screen the module needed in order to import a given predicate specification. For example:

```
?- where(findall/A).
findall/3 exported at module conc_aggregates
findall/4 exported at module aggregates
findall/3 exported at module aggregates

yes
?-
```

Usage: `where(Spec)`

– *Description:* Display what module to load in order to import the given `Spec`.

– *The following properties should hold at call time:*

`Spec` is a **Functor/Arity** predicate specification (librowser:pred_spec/1)

describe/1: PREDICATE

This one is used to find out which predicates were exported by a given module. Very usefull when you know the library, but not the concrete predicate. For example:

```
?- describe(librowser).
Predicates at library librowser :

apropos/1
system_lib/1
describe/1
where/1
browse/2
update/0

yes
?-
```

Usage: `describe(Module)`

- *Description:* Display a list of exported predicates at the given `Module`
- *The following properties should hold at call time:*

  `Module` is a module name (an atom) (librowser:module_name/1)

system_lib/1: PREDICATE

It retrieves on backtracking all Ciao system libraries stored in the internal database. Certainly, those which were scanned at `update/0` calling.

Usage: `system_lib(Module)`

- *Description:* `Module` variable will be successively instantiated to the system libaries stored in the internal database.
- *The following properties should hold at call time:*

  `Module` is a module name (an atom) (librowser:module_name/1)

apropos/1: PREDICATE

This tool makes use of regular expresions in order to find predicate specifications. It is very usefull whether you can't remember the full name of a predicate. Regular expresions take the same format as described in library `patterns`. Example:

```
?- apropos('atom_*').

terms: atom_concat/2
concurrency: atom_lock_state/2
atomic_basic: atom_concat/3,atom_length/2,atom_codes/2
iso_byte_char: atom_chars/2

yes
?-
```

Usage: `apropos(RegSpec)`

- *Description:* This will search any predicate specification `Spec` which matches the given `RegSpec` incomplete predicate specification.
- *The following properties should hold at call time:*

  `RegSpec` is a Pattern/Arity specification. (librowser:apropos_spec/1)

86.3 Documentation on internals (`librowser`)

apropos_spec/1: REGTYPE

Defined as:

```
apropos_spec(_1).
apropos_spec(_1/Arity) :-
        int(Arity).
```

Usage: `apropos_spec(S)`

– *Description:* `S` is a Pattern/Arity specification.

87 Code translation utilities

Author(s): Angel Fernandez Pineda.

This library offers a general way to perform clause body expansions. Goal, fact and spec translation predicates are authomatically called when needed, while this utility navigates through the meta-argument specification of the body itself. All predicates within this library must be called at *second-pass expansions*, since it uses information stored at `c_itf` library.

87.1 Usage and interface (`expansion_tools`)

- **Library usage:**
  This library is provided as a tool for those modules which performs source-to-source code translation, usually known as *code expanders*. It may be loaded as other modules using a `use_module/1`. Nothing special needs to be done.
- **Exports:**
  - *Predicates:*
    `imports_meta_pred/3`, `body_expander/6`, `arg_expander/6`.
- **Other modules used:**
  - *System library modules:*
    `compiler/c_itf`.

87.2 Documentation on exports (`expansion_tools`)

imports_meta_pred/3: PREDICATE

Macro provided in order to know meta-predicate specifications accessible from a module.

Usage: `imports_meta_pred(Module,MetaSpec,AccessibleAt)`

- *Description:* Tells whether `MetaSpec` meta-predicate specification is accessible from `Module`. `AccessibleAt` will be binded to '-' whether meta-predicate is a builtin one. If not, it will be unified with the module which defines the meta-predicate.
- *The following properties should hold at call time:*
  `Module` is an atom. (basic_props:atm/1)
  `MetaSpec` is any term. (basic_props:term/1)

body_expander/6: PREDICATE

This predicate is the main translation tool. It navigates through a clause body, when a single *goal* appears, user-code is called in order to perform a translation. Whether user-code fails to translate the involved goal, it remains the same. Regardless that goal is translated or not, an argument expansion will be performed over all goals if applicable (see `arg_expander/6` predicate).

Variable (unknown at compile time) goals will also be attempt to translate.

Meta-predicate with arguments: `body_expander((pred 3),(pred 3),(pred 3),?,?,?)`.

Usage:
`body_expander(GoalTrans,FactTrans,SpecTrans,Module,Body,ExpandedBody)`

– *Description:* Translates `Body` to `ExpandedBody` by the usage of user-defined translators `GoalTrans`, `FactTrans` and `SpecTrans`. The module where the original body appears must be unified with `Module` argument.
– *The following properties should hold at call time:*

`GoalTrans` is a user-defined predicate which performs *goal* meta-type translation (expansion_tools:goal_expander/1)

`FactTrans` is a user-defined predicate which performs *fact* meta-type translation (expansion_tools:fact_expander/1)

`SpecTrans` is a user-defined predicate which performs *spec* meta-type translation (expansion_tools:spec_expander/1)

`Module` is an atom. (basic_props:atm/1)

`ExpandedBody` is a free variable. (term_typing:var/1)

arg_expander/6: PREDICATE

This predicate is an auxiliary translation tool, which is used by `body_expander/6` predicate. It remains exported as a macro. The predicate navigates through the *meta-argument specification* of a goal. Whether a *goal,fact or spec* argument appears, user-code is called in order to perform a translation. Whether user-code fails to translate the involved argument, it remains the same. Builtins as ','/2 or ';'/2 are treated as meta-predicates defining *goal* meta-arguments. When a *goal* meta-argument is located, `body_expander/6` will be called in order to navigate through it. Notice that a *goal* meta-argument may be unified with another goal defining another meta-argument, so navigation is required. If arguments are not known to arg_expander/6, translation will not occur. This is posible whether goal or qualifing module are variables.

Meta-predicate with arguments: `arg_expander((pred 3),(pred 3),(pred 3),?,?,?).`

Usage:
`arg_expander(GoalTrans,FactTrans,SpecTrans,Module,Goal,ExpandedGoal)`

– *Description:* Translates `Goal` to `ExpandedGoal` by applying user-defined translators (`GoalTrans`, `FactTrans` and `SpecTrans`) to each meta-argument present at such goal. The module where the original goal appears must be unified with `Module` argument.
– *The following properties should hold at call time:*

`GoalTrans` is a user-defined predicate which performs *goal* meta-type translation (expansion_tools:goal_expander/1)

`FactTrans` is a user-defined predicate which performs *fact* meta-type translation (expansion_tools:fact_expander/1)

`SpecTrans` is a user-defined predicate which performs *spec* meta-type translation (expansion_tools:spec_expander/1)

`Module` is an atom. (basic_props:atm/1)

`ExpandedBody` is a free variable. (term_typing:var/1)

87.3 Documentation on internals (`expansion_tools`)

expander_pred/1: PROPERTY

Usage: `expander_pred(Pred)`

– *Description:* `Pred` is a user-defined predicate used to perform code translations. First argument will be binded to the corresponding term to be translated. Second argument must be binded to the corresponding translation. Third argument will be binded to the current module were first argument appears. Additional arguments will be user-defined.

87.4 Known bugs and planned improvements (`expansion_tools`)

- *pred(N)* meta-arguments are not supported at this moment.

88 Low-level concurrency/multithreading primitives

Author(s): Manuel Carro.

This module provides basic mechanisms for using concurrency and implementing multi-goal applications. It provides a means for arbitrary goals to be specified to be run in a separate stack set; in that case, they are assigned a goal identifier with which further accesses (e.g., asking for more solutions) to the goal can be made. Additionally, in some architectures, these goals can be assigned an O.S. thread, separate from the one which made the initial call, thus providing concurrency and, in multiprocessors, parallelism capabilities.

88.1 Usage and interface (concurrency)

- **Library usage:**
  `:- use_module(library(concurrency)).`
- **Exports:**
  - *Predicates:*
    `eng_call/4`, `eng_call/3`, `eng_backtrack/2`, `eng_cut/1`, `eng_release/1`, `eng_wait/1`, `eng_kill/1`, `eng_killothers/0`, `eng_self/1`, `goal_id/1`, `eng_goal_id/1`, `eng_status/0`, `lock_atom/1`, `unlock_atom/1`, `atom_lock_state/2`, `concurrent/1`.
- **Other modules used:**
  - *System library modules:*
    `foreign_interface/foreign_interface_properties`, `prolog_sys`.

88.2 Documentation on exports (concurrency)

eng_call/4: PREDICATE

Meta-predicate with arguments: `eng_call(goal,?,?,?)`.

Usage: `eng_call(Goal,EngineCreation,ThreadCreation,GoalId)`

- *Description:* Calls `Goal` in a new engine (stack set), possibly using a new thread, and returns a `GoalId` to designate this new goal henceforth. `EngineCreation` can be either `wait` or `create`; the distinction is not yet meaningful. `ThreadCreation` can be one of `self`, `wait`, or `create`. In the first case the creating thread is used to execute `Goal`, and thus it has to wait until its first result or failure. The call will fail if `Goal` fails, and succeed otherwise. However, the call will always suceed when a remote thread is started. The space and identifiers reclaimed for the thread must be explicitly deallocated by calling `eng_release/1`. `GoalId`s are unique in each execution of a Ciao Prolog program.
- *The following properties should hold at call time:*
  `Goal` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `EngineCreation` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `ThreadCreation` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `GoalId` is a free variable. (term_typing:var/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`EngineCreation` is an atom. (basic_props:atm/1)

`ThreadCreation` is an atom. (basic_props:atm/1)

`GoalId` is an integer. (basic_props:int/1)

eng_call/3: PREDICATE

Meta-predicate with arguments: `eng_call(goal,?,?)`.

Usage: `eng_call(Goal,EngineCreation,ThreadCreation)`

– *Description:* Similar to `eng_call/4`, but the thread (if created) and stack areas are automatically released upon success or failure of the goal. No `GoalId` is provided for further interaction with the goal.

– *The following properties should hold at call time:*

`Goal` is currently a term which is not a free variable. (term_typing:nonvar/1)

`EngineCreation` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ThreadCreation` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`EngineCreation` is an atom. (basic_props:atm/1)

`ThreadCreation` is an atom. (basic_props:atm/1)

eng_backtrack/2: PREDICATE

Usage: `eng_backtrack(GoalId,ThreadCreation)`

– *Description:* Performs backtracking on the goal designed by `GoalId`. A new thread can be used to perform backtracking, according to `ThreadCreation` (same as in `eng_call/4`). Fails if the goal is backtracked over by the local thread, and there are no more solutions. Always succeeds if executed by a remote thread. The engine is **not** automatically released up upon failure: `eng_release/1` must be called to that end.

– *The following properties should hold at call time:*

`GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ThreadCreation` is currently a term which is not a free variable. (term_typing:nonvar/1)

`GoalId` is an integer. (basic_props:int/1)

`ThreadCreation` is an atom. (basic_props:atm/1)

eng_cut/1: PREDICATE

Usage: `eng_cut(GoalId)`

– *Description:* Performs a *cut* in the execution of the goal `GoalId`. The next call to `eng_backtrack/2` will therefore backtrack all the way and fail.

– *The following properties should hold at call time:*

`GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)

`GoalId` is an integer. (basic_props:int/1)

eng_release/1: PREDICATE

Usage: `eng_release(GoalId)`

– *Description:* Cleans up and releases the engine executing the goal designed by `GoalId`. The engine must be idle, i.e., currently not executing any goal. `eng_wait/1` can be used to ensure this.

– *The following properties should hold at call time:*

`GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)

`GoalId` is an integer. (basic_props:int/1)

eng_wait/1: PREDICATE

Usage: `eng_wait(GoalId)`

– *Description:* Waits for the engine executing the goal denoted by `GoalId` to finish the computation (i.e., it has finished searching for a solution, either with success or failure).

– *The following properties should hold at call time:*

`GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)

`GoalId` is an integer. (basic_props:int/1)

eng_kill/1: PREDICATE

Usage: `eng_kill(GoalId)`

– *Description:* Kills the thread executing `GoalId` (if any), and frees the memory used up by the stack set. Usually one should wait (`eng_wait/1`) for a goal, and then release it, but killing the thread explicitly allows recovering from error states. A goal cannot kill itself. This feature should be used with caution, because there are situations where killing a thread might render the system in an unstable state. Threads should cooperate in their killing, but if the killed thread is blocked in a I/O operation, or inside an internal critical region, this cooperation is not possible and the system, although stopped, might very well end up in a incosistent state.

– *The following properties should hold at call time:*

`GoalId` is currently a term which is not a free variable. (term_typing:nonvar/1)

`GoalId` is an integer. (basic_props:int/1)

eng_killothers/0: PREDICATE

Usage:

– *Description:* Kills threads and releases stack sets of all active goals, but the one calling `eng_killothers`. Again, a safety measure. The same cautions as with `eng_kill/1` should be taken.

eng_self/1: PREDICATE

Usage: `eng_self(GoalId)`

– *Description:* `GoalId` is unified with the identifier of the goal within which `eng_self/1` is executed. `eng_self/1` is deprecated, and eng_goal_id/1 should be used instead.

– *The following properties should hold at call time:*

`GoalId` is an integer. (basic_props:int/1)

goal_id/1: PREDICATE

Usage: `goal_id(GoalId)`

- *Description:* `GoalId` is unified with the identifier of the goal within which `goal_id/1` is executed. `goal_id/1` is deprecated, and eng_goal_id/1 should be used instead.
- *The following properties should hold at call time:*

  `GoalId` is an integer. (basic_props:int/1)

eng_goal_id/1: PREDICATE

Usage: `eng_goal_id(GoalId)`

- *Description:* `GoalId` is unified with the identifier of the goal within which `eng_goal_id/1` is executed.
- *The following properties should hold at call time:*

  `GoalId` is an integer. (basic_props:int/1)

eng_status/0: PREDICATE

Usage:

- *Description:* Prints to standard output the current status of the stack sets.

lock_atom/1: PREDICATE

Usage 1:

- *The following properties should hold at call time:*

  `Arg1` is an integer. (basic_props:int/1)
- *The following properties hold globally:*

  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

Usage 2: `lock_atom(Atom)`

- *Description:* The semaphore associated to `Atom` is accessed; if its value is nonzero, it is atomically decremented and the execution of this thread proceeds. Otherwise, the goal waits until a nonzero value is reached. The semaphore is then atomically decremented and the execution of this thread proceeds.
- *The following properties should hold at call time:*

  `Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Atom` is an integer. (basic_props:int/1)

unlock_atom/1: PREDICATE

Usage 1:

- *The following properties should hold at call time:*

  `Arg1` is an integer. (basic_props:int/1)
- *The following properties hold globally:*

  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

Usage 2: `unlock_atom(Atom)`

– *Description:* The semaphore associated to `Atom` is atomically incremented.

– *The following properties should hold at call time:*

`Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Atom` is an integer. (basic_props:int/1)

atom_lock_state/2: PREDICATE

Usage 1:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

`Arg2` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

Usage 2: `atom_lock_state(Atom,Value)`

– *Description:* Sets the semaphore associated to `Atom` to `Value`. This is usually done at the beginning of the execution, but can be executed at any time. If not called, semaphore associated to atoms are by default inited to 1. It should be used with caution: arbitrary use can transform programs using locks in a mess of internal relations. The change of a semaphore value in a place other than the initialization stage of a program is **not** among the allowed operations as defined by Dijkstra [Dij65,BA82].

– *The following properties should hold at call time:*

`Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Value` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Atom` is an integer. (basic_props:int/1)

`Value` is an integer. (basic_props:int/1)

Usage 3: `atom_lock_state(Atom,Value)`

– *Description:* Consults the `Value` of the semaphore associated to `Atom`. Use sparingly and mainly as a medium to check state correctness. Not among the operations on semaphore by Djikstra.

– *The following properties should hold at call time:*

`Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Value` is a free variable. (term_typing:var/1)

`Atom` is an atom. (basic_props:atm/1)

`Value` is an integer. (basic_props:int/1)

concurrent/1: PREDICATE

`concurrent PredName`

The predicate named `PredName` is made concurrent in the current module at runtime (useful for predicate names generated on-the-fly). This difficults a better compile-time analysis, but in turn offers more flexibility to applications. It is also faster for some applications: if several agents have to share data in a stuctured fashion (e.g., the generator knows and wants to restrict the data generated to a set of other threads), a possibility

is to use the same concurrent fact and emply a field within the fact to distinguish the receiver/sender. This can cause many threads to access and wait on the same fact, which in turns can create contention problems. It is much better to create a new concurrent fact and to use that new name as a channel to communicate the different threads. `concurrent/1` can either be given a predicate spec in the form `Name/Arity`, with `Name` and `Arity` bound, or to give a value only to `Arity`, and let the system choose a new, unused `Name` for the fact.

Usage:

– *The following properties should hold at call time:*

`PredName` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

88.3 Known bugs and planned improvements (`concurrency`)

- Available only for Windows 32 environments and for architectures implementing POSIX threads.
- Some implementation of threads have a limit on the total number of threads that can be created by a process. Thread creation, in this case, just hangs. A better solution is planned for the future.

89 All solutions concurrent predicates

Author(s): Manuel Carro (concurrency-safeness).

This module implements thread-safe aggregation predicates. Its use and results should be the same as those in the aggregates library, but several goals can use them concurrently without the interference and wrong results (due to implementation reasons) aggregates might lead to. This particular implementation is completely based on the one used in the aggregates library (whose original authors were Richard A. O'Keefe and David H.D. Warren).

89.1 Usage and interface (conc_aggregates)

- **Library usage:**
  `:- use_module(library(conc_aggregates)).`
- **Exports:**
  - *Predicates:*
    `findall/3`, `setof/3`, `bagof/3`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`, `prolog_sys`.

89.2 Documentation on exports (conc_aggregates)

findall/3: PREDICATE

Meta-predicate with arguments: `findall(?,goal,?)`.

Usage: `findall(Template,Generator,List)` ⟨• ISO •⟩

- *Description:* A special case of bagof, where all free variables in the `Generator` are taken to be existentially quantified. Safe in concurrent applications.
- *The following properties should hold at call time:*
  `Generator` is currently a term which is not a free variable. (term_typing:nonvar/1)
- *The following properties should hold upon exit:*
  `Template` is any term. (basic_props:term/1)
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  `Set` is a list. (basic_props:list/1)
- *The following properties should hold globally:*
  All calls of the form `findall(Template,Generator,List)` are deterministic. (native_props:is_det/1)

setof/3: PREDICATE

Usage: `setof(Template,Goal,Set)` ⟨• ISO •⟩

– *Description:* Finds the `Set` of instances of the `Template` satisfying the `Generator`. The set is in ascending order (see `compare/3` for a definition of this order) without duplicates, and is non-empty. If there are no solutions, `setof/3` fails. `setof/3` may succeed in more than one way, binding free variables in the `Generator` to different values. This can be avoided by using existential quantifiers on the free variables in front of the `Generator`, using `^/2`. E.g., in `A^p(A,B)`, `A` is existentially quantified. Safe in concurrent apllications.

– *The following properties should hold at call time:*

`Goal` is currently a term which is not a free variable. (term_typing:nonvar/1)

– *The following properties should hold upon exit:*

`Template` is any term. (basic_props:term/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Set` is a list. (basic_props:list/1)

– *The following properties should hold globally:*

`Template` is not further instantiated. (basic_props:not_further_inst/2)

bagof/3: PREDICATE

Usage: `bagof(Template,Generator,Bag)` ⟨• ISO •⟩

– *Description:* Finds all the instances of the `Template` produced by the `Generator`, and returns them in the `Bag` in the order in which they were found. If the `Generator` contains free variables which are not bound in the `Template`, it assumes that this is like any other Prolog question and that you want bindings for those variables. This can be avoided by using existential quantifiers on the free variables in front of the `Generator`, using `^/2`. Safe in concurrent applications.

– *The following properties should hold at call time:*

`Generator` is currently a term which is not a free variable. (term_typing:nonvar/1)

– *The following properties should hold upon exit:*

`Template` is any term. (basic_props:term/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Set` is a list. (basic_props:list/1)

– *The following properties should hold globally:*

`Template` is not further instantiated. (basic_props:not_further_inst/2)

89.3 Known bugs and planned improvements (conc_aggregates)

- Thread-safe `setof/3` is not yet implemented.
- Thread-safe `bagof/3` is not yet implemented.

90 The socket interface

Author(s): Manuel Carro, Daniel Cabeza.

This module defines primitives to open sockets, send, and receive data from them. This allows communicating with other processes, on the same machine or across the Internet. The reader should also consult standard bibliography on the topic for a proper use of these primitives.

90.1 Usage and interface (sockets)

- **Library usage:**
  `:- use_module(library(sockets)).`
- **Exports:**
  - *Predicates:*
    `connect_to_socket_type/4`, `connect_to_socket/3`, `bind_socket/3`, `socket_accept/2`, `select_socket/5`, `socket_send/2`, `socket_recv_code/3`, `socket_recv/2`, `socket_shutdown/2`, `hostname_address/2`.
  - *Regular Types:*
    `socket_type/1`, `shutdown_type/1`.
- **Other modules used:**
  - *System library modules:*
    `foreign_interface/foreign_interface_properties`.

90.2 Documentation on exports (sockets)

connect_to_socket_type/4: PREDICATE

Usage: `connect_to_socket_type(Host,Port,Type,Stream)`

- *Description:* Returns a `Stream` which connects to `Host`. The `Type` of connection can be defined. A `Stream` is returned, which can be used to `write/2` to, to `read/2`, to `socket_send/2` to, or to `socket_recv/2` from the socket.
- *Calls should, and exit will be compatible with:*
  `Host` is currently instantiated to an atom. (term_typing:atom/1)
  `Port` is an integer. (basic_props:int/1)
  `Type` is a valid socket type. (sockets:socket_type/1)
  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties should hold at call time:*
  `Host` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Port` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Type` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Stream` is a free variable. (term_typing:var/1)
- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

connect_to_socket/3: PREDICATE

Usage: `connect_to_socket(Host,Port,Stream)`

- *Description:* Calls `connect_to_socket_type/4` with SOCK_STREAM connection type. This is the connection type you want in order to use the `write/2` and `read/2` predicates (and other stream IO related predicates).
- *Call and exit should be compatible with:*

  `Host` is an atom. (basic_props:atm/1)

  `Port` is an integer. (basic_props:int/1)

  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties should hold at call time:*

  `Host` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Port` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Stream` is a free variable. (term_typing:var/1)

bind_socket/3: PREDICATE

Usage: `bind_socket(Port,Length,Socket)`

- *Description:* Returs an AF_INET `Socket` bound to `Port` (which may be assigned by the OS or defined by the caller), and listens to it (hence no listen call in this set of primitives). `Length` specifies the maximum number of pending connections.
- *Calls should, and exit will be compatible with:*

  `Port` is an integer. (basic_props:int/1)

  `Length` is an integer. (basic_props:int/1)

  `Socket` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*

  `Length` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Socket` is a free variable. (term_typing:var/1)
- *The following properties hold globally:*

  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

socket_accept/2: PREDICATE

Usage: `socket_accept(Sock,Stream)`

- *Description:* Creates a new `Stream` connected to `Sock`.
- *Calls should, and exit will be compatible with:*

  `Sock` is an integer. (basic_props:int/1)

  `Stream` is an open stream. (streams_basic:stream/1)
- *The following properties should hold at call time:*

  `Sock` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Stream` is a free variable. (term_typing:var/1)
- *The following properties hold globally:*

  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

select_socket/5: PREDICATE

Usage: `select_socket(Socket,NewStream,TO_ms,Streams,ReadStreams)`

– *Description:* Wait for data available in a list of `Streams` and in a `Socket`. `Streams` is a list of Prolog streams which will be tested for reading. `Socket` is a socket (i.e., an integer denoting the O.S. port number) or a free variable. `TO_ms` is a number denoting a timeout. Within this timeout the `Streams` and the `Socket` are checked for the availability of data to be read. `ReadStreams` is the list of streams belonging to `Streams` which have data pending to be read. If `Socket` was a free variable, it is ignored, and `NewStream` is not checked. If `Socket` was instantiated to a port number and there are connections pending, a connection is accepted and connected with the Prolog stream in `NewStream`.

– *Calls should, and exit will be compatible with:*

`Socket` is an integer. (basic_props:int/1)

`NewStream` is an open stream. (streams_basic:stream/1)

`TO_ms` is an integer. (basic_props:int/1)

`Streams` is a list of `stream`s. (basic_props:list/2)

`ReadStreams` is a list of `stream`s. (basic_props:list/2)

– *The following properties should hold at call time:*

`Socket` is currently a term which is not a free variable. (term_typing:nonvar/1)

`NewStream` is a free variable. (term_typing:var/1)

`TO_ms` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Streams` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ReadStreams` is a free variable. (term_typing:var/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

socket_send/2: PREDICATE

Usage: `socket_send(Stream,String)`

– *Description:* Sends `String` to the socket associated to `Stream`. The socket has to be in connected state. `String` is not supposed to be NULL terminated, since it is a Prolog string. If a NULL terminated string is needed at the other side, it has to be explicitly created in Prolog.

– *Calls should, and exit will be compatible with:*

`Stream` is an open stream. (streams_basic:stream/1)

`String` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold at call time:*

`Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)

`String` is currently a term which is not a free variable. (term_typing:nonvar/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

socket_recv_code/3: PREDICATE

Usage: `socket_recv_code(Stream,String,Length)`

- *Description:* Receives a `String` from the socket associated to `Stream`, and returns its `Length`. If `Length` is -1, no more data is available.
- *Calls should, and exit will be compatible with:*

  `Stream` is an open stream. (streams_basic:stream/1)

  `String` is a string (a list of character codes). (basic_props:string/1)

  `Length` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*

  `Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)
- *The following properties hold globally:*

  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

socket_recv/2: PREDICATE

Usage: `socket_recv(Stream,String)`

- *Description:* As `socket_recv_code/3`, but the return code is ignored.
- *Call and exit should be compatible with:*

  `Stream` is an open stream. (streams_basic:stream/1)

  `String` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*

  `Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)

socket_shutdown/2: PREDICATE

Usage: `socket_shutdown(Stream,How)`

- *Description:* Shut down a duplex communication socket with which `Stream` is associated. All or part of the communication can be shutdown, depending on the value of `How`. The atoms `read`, `write`, or `read_write` should be used to denote the type of closing required.
- *Calls should, and exit will be compatible with:*

  `Stream` is an open stream. (streams_basic:stream/1)

  `How` is a valid shutdown type. (sockets:shutdown_type/1)
- *The following properties should hold at call time:*

  `Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `How` is currently a term which is not a free variable. (term_typing:nonvar/1)
- *The following properties hold globally:*

  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

hostname_address/2: PREDICATE

Usage: `hostname_address(Hostname,Address)`

– *Description:* `Address` is unified with the atom representing the address (in AF_INET format) corresponding to `Hostname`.

– *Calls should, and exit will be compatible with:*

`Hostname` is an atom. (basic_props:atm/1)

`Address` is an atom. (basic_props:atm/1)

– *The following properties should hold at call time:*

`Hostname` is currently a term which is not a free variable. (term_typing:nonvar/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

socket_type/1: REGTYPE

Defines the atoms which can be used to specify the socket type recognized by `connect_to_socket_type/4`. Defined as follows:

```
socket_type(stream).
socket_type(dgram).
socket_type(raw).
socket_type(seqpacket).
socket_type(rdm).
```

Usage: `socket_type(T)`

– *Description:* `T` is a valid socket type.

shutdown_type/1: REGTYPE

Usage: `shutdown_type(T)`

– *Description:* `T` is a valid shutdown type.

91 Sockets I/O

Author(s): Francisco Bueno.

This module provides two useful predicates for programming with sockets.

91.1 Usage and interface (sockets_io)

- **Library usage:**
  `:- use_module(library(sockets_io)).`
- **Exports:**
  - *Predicates:*
    `serve_socket/3`, `safe_write/2`.
- **Other modules used:**
  - *System library modules:*
    `lists`, `sockets/sockets`.

91.2 Documentation on exports (sockets_io)

serve_socket/3: PREDICATE

Meta-predicate with arguments: `serve_socket(?,(pred 1),(pred 1))`.

Usage: `serve_socket(Socket,Server,Handler)`

- *Description:* Handles the streams associated to `Socket` calling `Server` on one request of each stream (as `Server(Stream)`), and `Handler(Stream)` if the stream is empty (connection closed).
- *The following properties should hold at call time:*

  `Socket` is a socket id. (sockets_io:socket/1)

  `Server` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

  `Handler` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

safe_write/2: PREDICATE

Usage: `safe_write(Stream,Term)`

- *Description:* Writes `Term` to `Stream` in a way that it is safe for a socket connection on `Stream`.
- *The following properties should hold at call time:*

  `Stream` is an open stream. (streams_basic:stream/1)

  `Term` is any term. (basic_props:term/1)

92 The Ciao Make Package

Author(s): Manuel Hermenegildo.

This package is used mainly in two main ways:

- When writing `Makefile`s for `lpmake`.
- When writing *applications* which use the `make` library.

In both cases, this is the package that defines the syntax and meaning of the dependency rules used.

92.1 Usage and interface (`make_doc`)

- **Library usage:**
  - When writing `Makefile`s for `lpmake`, such makefiles start with:

    ```
    :- module(_,_,[make]).
    ```

    or

    ```
    :- make(_,_).
    ```

    (The latter uses the feature that an undefined declaration at the beginning of a file is interpreted by Ciao as a `use_module/3` including as third argument a package with the same name, in this case `make`.)
  - When writing *applications* which use the `make` package, then it is loaded as any other package within the application.

  Note: It is often useful to use the `fsyntax` package inside a `Makefile` (or when when using the `make` library in other applications). If both `make` and `fsyntax` are used, then `make` should appear **before** `fsyntax` in the list of packages.
- **New operators defined:**
  `::/2` [978,xfy], `<-/2` [977,xfy], `<=/2` [975,xfy], `<-/1` [977,yf].
- **Other modules used:**
  - *System library modules:*
    `make/make_rt`.

92.2 Other information (`make_doc`)

92.2.1 The Dependency Rules

The package allows defining the following types of rules:

TargetSuffix `<=` *SourceSuffix* `::` *SourceRoot* `:-` *BodyLiterals*`.`

A rule of this form declares that in order to produce the file with suffix *TargetSuffix* from a source file with the suffix *SourceSuffix* and root name *SourceRoot* the commands in *BodyLiterals* must be executed. *BodyLiterals* is a standard Ciao Prolog clause body, i.e., a comma-separated conjunction of literals. When writing the script, *SourceRoot* is typically left as a variable, to be instantiated by `lpmake` when the script is run to the root of name of the file to be processed. This allows using the value of *SourceRoot* in *BodyLiterals*. For example, the following rule:

```
:- use_module(library(terms), [atom_concat/2]).

dvi <= tex :: FileRoot :-
        atom_concat(['latex ',FileRoot,'.tex'],Command),
        system(Command).
```

states that we can generate a file *File*`.dvi` if we have a file named *File*`.tex` and that the command to do so is `latex` *File*`.tex`. Thus, if this rule appears in file `Makefile.pl` and we issue the command `lpmake paper.dvi` the following occurs:

- If `paper.dvi` does not exist and `paper.tex` exists, then `paper.dvi` is generated from `paper.tex` by issuing the system command `latex paper.tex`.
- If `paper.dvi` already exists, nothing is done.
- If `paper.tex` does not exist, an error is reported.

Target `<- :-` *BodyLiterals*`.`
A rule of this form declares that in order to produce the file *Target* the commands in *BodyLiterals* must be executed. *Target* need not be a real file: it can also be simply the name of the rule, which is used to invoke it (as a procedure name). For example, the following rule, when the command `lpmake realclean` is issued, deletes temporary files in the LaTeX application:

```
:- use_module(library(make(system_extra))).

clean <- :-
        ls('*aux|*log|*~',Files)
        delete_files(Files).
```

Target `<-` *Deps* `:-` *BodyLiterals*`.`
A rule of this form declares that in order to produce the file *Target*, first targets *Deps* will be called (i.e., the elements of *Deps* are either other targets with rules defined for them, or a file or files which are already present or which can –and will be– generated from other available files using other rules). Then, the commands in *BodyLiterals* will be executed. *Deps* may be one target or a list of targets. For example, the following rule, when the command `lpmake realclean` is issued, cleans all the temporary files in the LaTeX application (including `.dvi` and `.ps` files). It requires that clean be executed first:

```
:- use_package(fsyntax).
:- use_module(library(make(system_extra))).

realclean <- clean :-
        delete_files(~ls('*dvi|*ps')).
```

The following rule states that in order to meet the target `view`, target `paper.ps` must be available or generated. For example, `lpmake view` can be used to call the `ghostview` visualizer on `paper.ps`. Note the use of a globally defined *predicate* `main` which is called in two places in the rule, and could be used in other rules in the same file (`main := paper.` is equivalent to the fact `main(paper).` –see the `fsyntax` library):

```
:- use_package(fsyntax).
:- use_module(library(make(system_extra))).
:- use_module(library(terms), [atom_concat/2]).

main := paper.

view <- ~atom_concat([~main,'.ps']) :-
```

```
        system(~atom_concat(['ghostview ',~main,'.ps'])).
```

In addition to these rules, the configuration file can define normal predicates in the usual way, or import predicates from other modules, all of which can be called from the bodies of the dependency rules. For example, the `system_extra` library (an extension of the `system` library) defines many system predicates in a form which makes them very useful inside `Makefile`s, specially if the `fsyntax` package is used (see the examples below).

If `lpmake` is called without an explicit target as argument, then the first target rule in the Makefile is used. This is useful in that the first rule can be seen as the default rule.

92.2.2 Specifying Paths

Using the `vpath/1` predicate it is possible in configuration files to define several paths in which files related to the rules can be located. In this way, not all files need to be in the same directory as the configuration file. For example:

```
:- use_package(fsyntax).

vpath := '/home/clip/Systems/ciao/lib'.
vpath := '/home/clip/Systems/ciao/library'.
vpath := '/home/clip/Systems/lpdoc/lib'.
```

92.2.3 Documenting Rules

It is also possible to define documentation for the rules:

`target_comment(`*Target*`) :-` *BodyLiterals*`.`
: A rule of this form allows documenting the actions related to the target. The body (*BodyLiterals*) will be called in two circumstances:

  - If *Target* is called during execution of '`lpmake` *commands*'.
  - When calling '`lpmake -h`'.

  Using noun forms (*generation of foo* instead of *generating foo*) in comments helps this dual purpose. For example, the following rule:

  ```
  target_comment(realclean) :-
          display('Cleanup of all generated files.').
  ```

  will produce output in the two cases pointed out above.

`dependency_comment(`*SourceSuffix*`, `*TargetSuffix*`, `*SourceRoot*`) :-` *BodyLiterals*`.`
: Same as the previous rule, but for suffix rules. See, for example, the following generic rule:

  ```
  :- use_module(library(terms), [atom_concat/2]).

  dependency_comment(SSuffix,TSuffix,FileBase) :-
          display(~atom_concat(['Generation of ',FileBase,'.',
          TSuffix, ' from ',FileBase,'.',SSuffix])).
  ```

92.2.4 An Example of a Makefile

The following is a simple example of a Makefile showing some basic functionality (this is `MakefileExample.pl` in the `example_simple` directory in the `make` library.):

```
%% ---------------------------------------------------------------------
:- module(_,_,[make,fsyntax]).
:- use_module(library(make(system_extra))).
```

```
:- use_module(library(lists), [append/3]).
:- use_module(library(terms), [atom_concat/2]).

:- discontiguous(comment/2).

%% ---------------------------------------------------------------------
%% A simple target. Defines how to produce file 'hw'.

hw <-  []    :-
         writef("Hello world", hw).

%% A comment describing this target (see below):
comment(hw,['Generation of file hw']).

%% ---------------------------------------------------------------------
%% A target with a dependency. 'hwhw' requires 'hw'.

hwhw <- [hw] :-
        readf(hw,Content),
        append(Content,[0'\n|Content],DoubleContent),
        writef(DoubleContent,hwhw).

comment(hwhw,['Generation of file hwhw']).

%% ---------------------------------------------------------------------
%% A simple target. Defines how to produce file 'datafile.simple'.

'datafile.simple' <-  :-
        writef("Hello world", 'datafile.simple').

comment('datafile.simple',['Generation of file datafile.simple']).

%% ---------------------------------------------------------------------
%% A dependency based on suffixes:
%% <file>.double is generated always from <file>.simple

double <= simple :: Name :-
        readf(~atom_concat([Name,'.simple']),Content),
        append(Content,[0'\n|Content],DoubleContent),
        writef(DoubleContent,~atom_concat([Name,'.double'])).

%% ---------------------------------------------------------------------
%% A dependency based on suffixes with a precondition.
%% <file>.double is generated always from <file>.simple, once
%% precond is done

boo <- :-
        display((double <= simple :: name <- precond :- body1, body2)).

%% ---------------------------------------------------------------------
%% Example using library predicates
```

```
clean <- [] # "Cleanup of temporary files " :-
        delete_files(~ls('*~|*.asr|*.itf|*.po')).

realclean <- clean :-
        delete_files(~ls('hw|hwhw|*simple|*double')).

comment(realclean,['Cleanup of all generated files']).

%% ---------------------------------------------------------------------------
%% Reporting progress and documenting commands:
%% If target_comment/1 is defined it can be used to produce user-defined
%% output when targets are processed and/or documentation on what each
%% target does (used for example when lpmake is called with -h). Using
%% 'generation of foo' instead of 'generating foo' in comments helps in this
%% dual purpose.
%% ---------------------------------------------------------------------------

:- push_prolog_flag(multi_arity_warnings,off).
%% Make calls target_comment/1 for simple targets:
target_comment(Target) :-
        comment(Target,Comment),
        display(~atom_concat([~atom_concat(Comment), '\n'])).
:- pop_prolog_flag(multi_arity_warnings).

%% Similarly, make calls dependency_comment/3 for dependencies (only
%% during execution, not when documenting -h).
dependency_comment(SSuffix,TSuffix,FileBase) :-
        display(~atom_concat(['Generation of ',FileBase,TSuffix,
                             ' from ',FileBase,SSuffix,'\nl'])).
```

The following are a few commands that can be used on the previous file (see file `CommandsToTry` in the `example_simple` directory in the `make` library):

```
lpmake -m MakefileExample.pl hwhw
(Generate file hwhw --needs to generate file hw first)

lpmake -m MakefileExample.pl datafile.double
(Generate file datafile.double --needs to generate file
datafile.simple first)

lpmake -m MakefileExample.pl realclean
(Cleanup)

lpmake -h -m MakefileExample.pl
(Help on general use of lpmake and commands available in MakefileExample.pl)
```

See also the LaTeX example in the `example_latex` directory in the `make` library.

93 Predicates Available When Using The Make Package

Author(s): Manuel Hermenegildo, Edison Mera.

This is the run-time module which implements the predicates which are provided when using the `make` library package in a given application. For example, they are used internally by `lpmake`.

93.1 Usage and interface (`make_rt`)

- **Library usage:**
  This module is loaded automatically when the `make` library package is used.
- **Exports:**
  - *Predicates:*
    `make/1`, `make_option/1`, `verbose_message/1`, `verbose_message/2`, `dot_concat/2`, `call_unknown/1`, `all_values/2`, `get_value/2`, `get_value_def/3`, `get_all_values/2`, `name_value/2`, `set_name_value/2`, `cp_name_value/2`, `get_name_value/3`, `get_name_value_string/3`, `add_name_value/2`, `del_name_value/1`, `check_var_exists/1`, `find_file/2`, `vpath/1`, `add_vpath/1`, `vpath_mode/3`, `add_vpath_mode/3`, `bold_message/1`, `bold_message/2`, `normal_message/2`, `bolder_message/1`, `bolder_message/2`, `newer/2`, `register_module/1`, `unregister_module/1`, `push_name_value/3`, `pop_name_value/1`, `push_active_config/1`, `pop_active_config/0`, `get_active_config/1`, `dyn_load_cfg_module_into_make/1`, `get_settings_nvalue/1`, `apply_vpath_mode/4`, `get_name/2`.
  - *Regular Types:*
    `target/1`.
- **Other modules used:**
  - *System library modules:*
    `compiler/compiler`, `filenames`, `terms`, `system`, `format`, `lists`, `messages`, `make/up_to_date`, `aggregates`.

93.2 Documentation on exports (`make_rt`)

make/1: PREDICATE

Usage: `make(TargetList)`

- *Description:* This is the main entry point to the make library. It makes the list of targets one by one as well as any intermediate targets needed as dictated by the dependency rules.
- *The following properties should hold at call time:*
  `TargetList` is a list of `target`s. (basic_props:list/2)

target/1: REGTYPE

Usage: `target(T)`

- *Description:* `T` is a Makefile target.

make_option/1: PREDICATE

The predicate is of type *data*.

Usage: `make_option(Option)`

– *Description:* Asserting/retracting facts of this predicate sets/clears library options. Default is no options (i.e., the predicate is undefined). The following values are supported:

```
make_option('-v'). % Verbose: prints progress messages (useful
                   % for debugging rules).
```

– *The following properties should hold at call time:*

`Option` is an atom. (basic_props:atm/1)

verbose_message/1: PREDICATE

No further documentation available for this predicate.

verbose_message/2: PREDICATE

Usage: `verbose_message(Text,ArgList)`

– *Description:* The text provided in `Text` is printed as a message, using the arguments in `ArgList`, if `make_option('-v')` is defined. Otherwise nothing is printed.

– *The following properties should hold at call time:*

`Text` is an atom or string describing how the arguments should be formatted. If it is an atom it will be converted into a string with `name/2`. (format:format_control/1)

`ArgList` is a list. (basic_props:list/1)

dot_concat/2: PREDICATE

No further documentation available for this predicate.

call_unknown/1: PREDICATE

No further documentation available for this predicate.

all_values/2: PREDICATE

No further documentation available for this predicate.

get_value/2: PREDICATE

No further documentation available for this predicate.

get_value_def/3: PREDICATE

No further documentation available for this predicate.

get_all_values/2: PREDICATE

Usage: `get_all_values(Name,Values)`

– *Description:* `Values` are all the possible values of `Name`.

name_value/2: PREDICATE

No further documentation available for this predicate.

The predicate is of type *data*.

set_name_value/2: PREDICATE

No further documentation available for this predicate.

cp_name_value/2: PREDICATE

Usage: `cp_name_value(Source,Target)`

– *Description:* Copy the variable values from `Source` to `Target`

– *The following properties should hold at call time:*

`Source` is an atom. (basic_props:atm/1)

`Target` is an atom. (basic_props:atm/1)

get_name_value/3: PREDICATE

No further documentation available for this predicate.

get_name_value_string/3: PREDICATE

No further documentation available for this predicate.

add_name_value/2: PREDICATE

No further documentation available for this predicate.

del_name_value/1: PREDICATE

No further documentation available for this predicate.

check_var_exists/1: PREDICATE

Usage: `check_var_exists(Var)`

– *Description:* Fails printing a message if variable `Var` does not exist.

find_file/2: PREDICATE

No further documentation available for this predicate.

vpath/1: PREDICATE
No further documentation available for this predicate.
The predicate is of type *data*.

add_vpath/1: PREDICATE
No further documentation available for this predicate.

vpath_mode/3: PREDICATE
No further documentation available for this predicate.
The predicate is of type *data*.

add_vpath_mode/3: PREDICATE
No further documentation available for this predicate.

bold_message/1: PREDICATE
No further documentation available for this predicate.

bold_message/2: PREDICATE
No further documentation available for this predicate.

normal_message/2: PREDICATE
No further documentation available for this predicate.

bolder_message/1: PREDICATE
No further documentation available for this predicate.

bolder_message/2: PREDICATE
No further documentation available for this predicate.

newer/2: PREDICATE
No further documentation available for this predicate.

register_module/1: PREDICATE
No further documentation available for this predicate.

unregister_module/1: PREDICATE

No further documentation available for this predicate.

push_name_value/3: PREDICATE

Usage: `push_name_value(Name,Var,R)`

- *Description:* Push variable name `Name` with all values of variable `Var` and returns `R`, an abstract type to pass to `pop_name_value/1` to undo the push changes. Push cannot be nested.

pop_name_value/1: PREDICATE

Usage: `pop_name_value(R)`

- *Description:* Restores the value of the variable indicated by the abstract type `R`. Notice that `R` _must be_ the argument returned by `push_name_value/2`.

push_active_config/1: PREDICATE

No further documentation available for this predicate.

pop_active_config/0: PREDICATE

No further documentation available for this predicate.

get_active_config/1: PREDICATE

No further documentation available for this predicate.

dyn_load_cfg_module_into_make/1: PREDICATE

Usage: `dyn_load_cfg_module_into_make(ConfigFile)`

- *Description:* Used to load dynamically a module (typically, a `Makefile`) into the make library from the application using the library.
- *The following properties should hold at call time:*

  `ConfigFile` is a source name. (streams_basic:sourcename/1)

get_settings_nvalue/1: PREDICATE

Usage: `get_settings_nvalue(Pred)`

- *Description:* Executes `Pred` as unkown call, in other words, it is useful to execute predicates that have been loaded by `register_module/1`.

  Example: get_settings_nvalue(component_options(ciao, A, B)).
- *The following properties should hold at call time:*

  `Pred` is any term. (basic_props:term/1)

apply_vpath_mode/4: PREDICATE

No further documentation available for this predicate.

get_name/2: PREDICATE

Usage: `get_name(Path,Name)`

– *Description:* `Name` is the file name for the path `Path`

up_to_date/2: (UNDOC_REEXPORT)

Imported from `up_to_date` (see the corresponding documentation for details).

93.3 Known bugs and planned improvements (`make_rt`)

- The current handle of help messages is defficient. It must be in a standar form, and the user of this library only must be allowed to add messages, not procedures to print it.
- target_comment/1 does not work, why? :-(.

94 Additional operating system utilities

Author(s): Manuel Hermenegildo.

This is a (temporary) extension to library `system` (which it reexports). It implements functionality that is often convenient in `Makefile`s. Much of this should probably end up eventually in `system`, but once we have worked out the best interface and, in some cases, the proper implementation (the implementations in here are in some cases just calls to Un*x shell primitives or commands).

94.1 Usage and interface (system_extra)

- **Library usage:**
  `:- use_module(library(system_extra)).`
- **Exports:**
  - *Predicates:*
    `del_dir_if_empty/1`, `move_files/2`, `move_file/2`, `copy_files/2`, `copy_files/3`, `copy_files_nofail/3`, `cat/2`, `cat_append/2`, `symbolic_link/2`, `symbolic_link/3`, `delete_files/1`, `del_files_nofail/1`, `del_file_nofail/1`, `del_file_nofail/2`, `del_endings_nofail/2`, `ls/3`, `ls/2`, `filter_alist_pattern/3`, `--/1`, `do/2`, `do/3`, `do/4`, `do/5`, `try_finally/3`, `set_owner/2`, `readf/2`, `datime_string/1`, `datime_string/2`, `datime_atom/1`, `datime_atom/2`, `no_tr_nl/2`, `replace_strings/3`, `replace_strings_in_file/3`, `writef/3`, `writef/2`, `add_suffix/3`, `add_preffix/3`, `writef_list/3`, `writef_list/2`, `etags/2`, `any_to_term/2`, `touch/1`, `get_perms/2`, `set_perms/2`, `set_exec_perms/2`, `mkdir_perm/2`, `convert_permissions/2`, `convert_permissions/4`, `execute_permissions/2`, `execute_permissions/4`, `do_str/3`, `do_str_without_nl/3`, `do_str_without_nl__popen/2`, `do_atmlist__popen/2`.
  - *Regular Types:*
    `pattern/1`, `do_options/1`.
- **Other modules used:**
  - *System library modules:*
    `regexp/regexp_code`, `system`, `messages`, `terms`, `lists`, `llists`, `sort`, `read`, `write`, `file_utils`, `strings`.

94.2 Documentation on exports (system_extra)

del_dir_if_empty/1: PREDICATE

No further documentation available for this predicate.

move_files/2: PREDICATE

`move_files(Files,Dir)`

Move `Files` to directory `Dir` (note that to move only one file to a directory, `rename_file/2` can be used).

Usage:

– *The following properties should hold at call time:*
`Files` is a list of `atm`s. (basic_props:list/2)
`Dir` is an atom. (basic_props:atm/1)

move_file/2: PREDICATE
No further documentation available for this predicate.

copy_files/2: PREDICATE
`copy_files(Files,Dir)`
Copy `Files` to directory `Dir` (note that to move only one file to a directory, `rename_file/2` can be used).
Usage:
– *The following properties should hold at call time:*
`Files` is a list of `atm`s. (basic_props:list/2)
`Dir` is an atom. (basic_props:atm/1)

copy_files/3: PREDICATE
No further documentation available for this predicate.

copy_files_nofail/3: PREDICATE
No further documentation available for this predicate.

cat/2: PREDICATE
No further documentation available for this predicate.

cat_append/2: PREDICATE
No further documentation available for this predicate.

symbolic_link/2: PREDICATE
Usage: `symbolic_link(Source,Dir)`
– *Description:* Create a symbolic link in `Dir` pointing to file or directory `Source` (performs a copy in Windows).

symbolic_link/3: PREDICATE
Usage: `symbolic_link(Source,Dir,NewName)`
– *Description:* Create a symbolic link in `Dir` pointing to file or directory `Source` and give it name `NewName` (performs a copy in Windows).

delete_files/1: PREDICATE

No further documentation available for this predicate.

del_files_nofail/1: PREDICATE

No further documentation available for this predicate.

del_file_nofail/1: PREDICATE

No further documentation available for this predicate.

del_file_nofail/2: PREDICATE

No further documentation available for this predicate.

del_endings_nofail/2: PREDICATE

No further documentation available for this predicate.

ls/3: PREDICATE

`ls(Directory,Pattern,FileList)`

`FileList` is the unordered list of entries (files, directories, etc.) in `Directory` whose names match `Pattern`.If `Directory` does not exist `FileList` is empty.

Usage:

- *The following properties should hold at call time:*

  `Directory` is an atom. (basic_props:atm/1)

  system_extra:pattern(Pattern) (system_extra:pattern/1)

  `FileList` is a free variable. (term_typing:var/1)

- *The following properties hold upon exit:*

  `FileList` is a list of `atm`s. (basic_props:list/2)

ls/2: PREDICATE

`ls(Pattern,FileList)`

`FileList` is the unordered list of entries (files, directories, etc.) in the current directory whose names match `Pattern` (same as `ls('.',Pattern,FileList)`).

Usage:

- *The following properties should hold at call time:*

  system_extra:pattern(Pattern) (system_extra:pattern/1)

  `FileList` is a free variable. (term_typing:var/1)

- *The following properties hold upon exit:*

  `FileList` is a list of `atm`s. (basic_props:list/2)

filter_alist_pattern/3: PREDICATE

`filter_alist_pattern(UnFiltered,Pattern,Filtered)`

`Filtered` contains the elements of `UnFiltered` which match with `Pattern`.

Usage:

- *The following properties should hold at call time:*
  `UnFiltered` is a list of `atm`s. (basic_props:list/2)
  system_extra:pattern(Pattern) (system_extra:pattern/1)
  `Filtered` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*
  `Filtered` is a list of `atm`s. (basic_props:list/2)

-/1: PREDICATE

No further documentation available for this predicate.

Meta-predicate with arguments: `-goal`.

–/1: PREDICATE

No further documentation available for this predicate.

Meta-predicate with arguments: `--goal`.

do/2: PREDICATE

Usage: `do(Command,Action)`

- *Description:* Same as `do/3` but omitting the return code.
- *The following properties should hold at call time:*
  `Command` is a list of `atm`s. (basic_props:list/2)
  `Action` is a list of `do_option`s. (basic_props:list/2)

do/3: PREDICATE

Usage: `do(Command,Action,ReturnCode)`

- *Description:* Same as `do/5` but omitting the files.
- *The following properties should hold at call time:*
  `Command` is a list of `atm`s. (basic_props:list/2)
  `Action` is a list of `do_option`s. (basic_props:list/2)
- *The following properties should hold upon exit:*
  `ReturnCode` is a number. (basic_props:num/1)

do/4: PREDICATE

Usage: `do(Command,OutputFile,ErrorFile,Action)`

- *Description:* Same as `do/5` but omitting the returned code.

– *The following properties should hold at call time:*

`Command` is a list of `atm`s. (basic_props:list/2)

`OutputFile` is an atom. (basic_props:atm/1)

`ErrorFile` is an atom. (basic_props:atm/1)

`Action` is a list of `do_option`ss. (basic_props:list/2)

do/5: PREDICATE

Usage: `do(Command,OutputFile,ErrorFile,Action,ReturnCode)`

– *Description:* Executes `Command` redirecting standard output to `OutputFile` and standard error to `ErrorFile`. `ReturnCode` is the code returned by the execution of `Command`. `Action` is a list of atoms that specify the actions to be completed in case the `Command` fails. Three of these options: `fail`, `exception`, and `nofail` are mutually exclusive. The rest of the options are flags that mean (type `do_options/1`):

`inform_nofail`: informs about the error code returned by the execution of the command.

`show_output_on_error`: shows the content of `OutputFile` in case of error.

`show_error`: shows the content of `ErrorFile` in case of error.

`silent`: do not print any error message. The option `inform_nofail` overrides this option in case of `fail`.

`verbose_command`: shows the command before being executed. Useful for tracing.

`verbose`: `verbose_command` plus overrides the error and output file settings and outputs everything to `user_output` and `user_error`.

– *The following properties should hold at call time:*

`Command` is a list of `atm`s. (basic_props:list/2)

`OutputFile` is an atom. (basic_props:atm/1)

`ErrorFile` is an atom. (basic_props:atm/1)

`Action` is a list of `do_option`ss. (basic_props:list/2)

– *The following properties should hold upon exit:*

`ReturnCode` is a number. (basic_props:num/1)

try_finally/3: PREDICATE

Meta-predicate with arguments: `try_finally(goal,goal,goal)`.

Usage: `try_finally(Start,Goal,End)`

– *Description:* Calls initialization goal `Start` and then calls Goal `Goal`, but always continues with the evaluation of `End`. If `Goal` is non-deterministic, in case of backtracking `Start` is called again before redoing `Goal`.

set_owner/2: PREDICATE

No further documentation available for this predicate.

readf/2: PREDICATE

No further documentation available for this predicate.

datime_string/1: PREDICATE
No further documentation available for this predicate.

datime_string/2: PREDICATE
No further documentation available for this predicate.

datime_atom/1: PREDICATE
No further documentation available for this predicate.

datime_atom/2: PREDICATE
No further documentation available for this predicate.

no_tr_nl/2: PREDICATE
No further documentation available for this predicate.

replace_strings/3: PREDICATE
No further documentation available for this predicate.

replace_strings_in_file/3: PREDICATE
No further documentation available for this predicate.

writef/3: PREDICATE
No further documentation available for this predicate.

writef/2: PREDICATE
No further documentation available for this predicate.

add_suffix/3: PREDICATE
No further documentation available for this predicate.

add_preffix/3: PREDICATE
No further documentation available for this predicate.

writef_list/3: PREDICATE
No further documentation available for this predicate.

writef_list/2: PREDICATE
No further documentation available for this predicate.

etags/2: PREDICATE
No further documentation available for this predicate.

any_to_term/2: PREDICATE
`any_to_term(Any,Term)`
Interprets the result of print `Any` as a term `Term`

touch/1: PREDICATE
`touch(File)`
Updates the access and modification time of `File` to current time.

get_perms/2: PREDICATE
No further documentation available for this predicate.

set_perms/2: PREDICATE
No further documentation available for this predicate.

set_exec_perms/2: PREDICATE
No further documentation available for this predicate.

mkdir_perm/2: PREDICATE
No further documentation available for this predicate.

convert_permissions/2: PREDICATE
No further documentation available for this predicate.

convert_permissions/4: PREDICATE
No further documentation available for this predicate.

execute_permissions/2: PREDICATE
No further documentation available for this predicate.

execute_permissions/4: PREDICATE

No further documentation available for this predicate.

pattern/1: REGTYPE

A regular type, defined as follows:

```
pattern(A) :-
        atom(A).
```

do_options/1: REGTYPE

A regular type, defined as follows:

```
do_options(fail).
do_options(nofail).
do_options(silent).
do_options(exception).
do_options(halt).
do_options(inform_nofail).
do_options(show_output_on_error).
do_options(show_error_on_error).
do_options(verbose).
do_options(verbose_command).
```

do_str/3: PREDICATE

No further documentation available for this predicate.

do_str_without_nl/3: PREDICATE

No further documentation available for this predicate.

do_str_without_nl__popen/2: PREDICATE

No further documentation available for this predicate.

do_atmlist__popen/2: PREDICATE

No further documentation available for this predicate.

system_error_report/1: (UNDOC_REEXPORT)

Imported from `system` (see the corresponding documentation for details).

replace_characters/4: (UNDOC_REEXPORT)

Imported from `system` (see the corresponding documentation for details).

no_swapslash/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

cyg2win/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

winpath_c/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

winpath/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

winpath/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

using_windows/0: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

rename_file/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

delete_directory/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

delete_file/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

set_exec_mode/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

chmod/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

chmod/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

fmode/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

modif_time0/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

modif_time/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_properties/6: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_property/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_exists/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_exists/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

mktemp_in_tmp/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

mktemp/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

directory_files/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

wait/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

exec/8: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

exec/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

exec/4: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

popen_mode/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

popen/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

system/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

system/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

shell/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

shell/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

shell/0: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

cd/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

working_directory/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

make_dirpath/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

make_dirpath/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

make_directory/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

make_directory/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

umask/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

current_executable/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

current_host/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_address/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_tmp_dir/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_grnam/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_pwnam/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_gid/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_uid/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

get_pid/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

file_dir_name/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

extract_paths/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

dir_path/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

copy_file/3: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

copy_file/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

c_errno/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

del_env/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

set_env/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

current_env/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

setenvstr/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

getenvstr/2: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

datime_struct/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

datime/9: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

datime/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

time/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

pause/1: (UNDOC_REEXPORT)
Imported from `system` (see the corresponding documentation for details).

PART VII - Ciao extensions

Author(s): The CLIP Group.

The libraries documented in this part extend the Ciao language in several different ways. The extensions include:

- pure Prolog programming (well, this can be viewed more as a restriction than an extension);
- feature terms or *records* (i.e., structures with names for each field);
- parallel programming (e.g., &-Prolog style);
- functional syntax;
- higher-order;
- global variables;
- `setarg` and `undo`;
- delaying predicate execution;
- active modules;
- breadth-first execution;
- iterative deepening-based execution;
- constraint logic programming;
- object oriented programming.

95 Pure Prolog package

Author(s): The CLIP Group.

This library package allows the use of *pure Prolog* in a Ciao module/program. It is based on the fact that if an *engine module* is imported explicitly then all of them have to be imported explicitly. The engine modules are:

- `engine(arithmetic)`
  Chapter 20 [Arithmetic], page 151.
- `engine(atomic_basic)`
  Chapter 19 [Basic predicates handling names of constants], page 141.
- `engine(attributes)`
  Chapter 28 [Attributed variables], page 201.
- `engine(basic_props)`
  Chapter 15 [Basic data types and properties], page 105.
- `engine(basiccontrol)`
  Chapter 13 [Control constructs/predicates], page 99.
- `engine(data_facts)`
  Chapter 25 [Fast/concurrent update of facts], page 187.
- `engine(exceptions)`
  Chapter 23 [Exception and Signal handling], page 177.
- `engine(io_aux)`
  Chapter 27 [Message printing primitives], page 197.
- `engine(io_basic)`
  Chapter 22 [Basic input/output], page 169.
- `engine(prolog_flags)`
  Chapter 24 [Changing system behaviour and various flags], page 181.
- `engine(streams_basic)`
  Chapter 21 [Basic file/stream handling], page 159.
- `engine(system_info)`
  ⟨undefined⟩ [Gathering some basic internal info], page ⟨undefined⟩.
- `engine(term_basic)`
  Chapter 17 [Basic term manipulation], page 129.
- `engine(term_compare)`
  Chapter 18 [Comparing terms], page 135.
- `engine(term_typing)`
  Chapter 16 [Extra-logical properties for typing], page 121.

Note that if any of these modules is explicitely imported in a program then the language defaults to Pure Prolog, plus the functionality added by the modules explicitely imported.

It is recommended that if you explicitely import an engine module you also use this package, which will guarantee that the predicate `true/0` is defined (note that this is the only Ciao builtin which cannot be redefined).

95.1 Usage and interface (pure_doc)

- **Library usage:**
  `:- use_package(pure).`
  or
  `:- module(...,...,[pure]).`

95.2 Known bugs and planned improvements (pure_doc)

- Currently, the following builtin predicates/program constructs cannot be redefined, in addition to `true/0`: `(->)/2` `(,)/2` `(+)/1` `if/3`

96 Multiple Argument Indexing

Author(s): Anil Nair (original work), Tom Howland (`http://home.pacbell.net/tomjdnh/pd.html`, derived the original work), Francisco Bueno (port to Ciao).

This package is an extension of the idea of Prolog indexing, usually performed, in a limited way, on the first argument. This package provides more powerful indexing schemes. It lets you pick different arguments to index on, and provides for different combinations of arguments to index on. E.g., it will let you index on the first and third argument or the second and the third argument of a predicate.

The indexing is based on computing a hash value for the terms to be indexed upon. Note, however, that the current implementation of the package is done at the source level, so it may sometimes not be as fast as expected. Given this, this version of the package pays off only when the amount of clashing that your original predicate causes without the package superseeds the cost of the hashing function in the package. Such amount of course depends on the number and the form of the facts in your predicate.

96.1 Usage and interface (`indexer_doc`)

- **Library usage:**
  This facility is used as a package, thus either including `indexer` in the package list of the module, or by using the `use_package/1` declaration. The facility predicate `hash_term/2`, documented here, is defined in library module `indexer(hash)`.
- **Exports:**
  - *Predicates:*
    `hash_term/2`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`, `indexer/hash`.

96.2 Documentation on exports (`indexer_doc`)

hash_term/2: PREDICATE

`hash_term(Term,HashValue)`

Provides an efficient way to calculate an integer `HashValue` for a ground `Term`.

Usage 1: `hash_term(T,N)`

- *Description:* `N` is a hashing index for `T`.
- *The following properties should hold at call time:*
  `T` is currently ground (it contains no variables). (term_typing:ground/1)
  `N` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  `N` is an integer. (basic_props:int/1)

Usage 2: `hash_term(T,N)`

– *The following properties should hold at call time:*

`T` is not ground. (native_props:nonground/1)

`N` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`N` is a free variable. (term_typing:var/1)

96.3 Documentation on internals (`indexer_doc`)

index/1: DECLARATION

Usage: `:- index(IndexSpecs).`

– *Description:* Declares an indexing scheme for a predicate. All specs of `IndexSpecs` must be terms for the same predicate. Each spec declares an indexing on a combination of the arguments. Indexing will be performed using any of the specs in `IndexSpecs` (being thus interpreted as an or).

You should use a `*` in an argument position if you wish to hash on the entire term in that argument. If a `+` is used only one level of the term in the argument is used for hashing. An `i` is used to indicate that argument is already an integer, and therefore its own value will be used for hashing. The argspec `?` simply indicates not to use the argument for indexing.

For example, the index specification:

```
:- index foo(+,?,*,i), foo(?,?,?,i).
```

declares indexing for `foo/4` either on a combination of the first, third, and fourht arguments, or only on the last argument, which is an integer. In the first case, only the principal functor of the first argument will be used for hashing; the third argument will be used in its entirety.

The argspec `n` is a pragmatic extension and can not be used in conjunction with the other specifiers aside from `?`. It stands for "nonvar" and implies that the argument will not be used for hashing, since only ground terms can effectively be used in hashing. Thus, it can not be used in combination with other specifiers within a particular index specification. It is often the fastest thing to use.

– *The following properties should hold upon exit:*

`IndexSpecs` is an index specification. (indexer_doc:indexspecs/1)

indexspecs/1: REGTYPE

An index specification is defined as follows:

```
indexspecs(Spec) :-
        indexspec(Spec).
indexspecs((Spec,Specs)) :-
        indexspec(Spec),
        indexspecs(Specs).

indexspec(Spec) :-
        Spec=..[_F|Args],
        list(Args,argspec).
```

Usage: `indexspecs(IndexSpecs)`

– *Description:* `IndexSpecs` is an index specification.

argspec/1: REGTYPE

An argument hash specification is defined as follows:

```
argspec(+).
argspec(*).
argspec(i).
argspec(n).
argspec(?).
```

Usage: `argspec(Spec)`

– *Description:* `Spec` is an argument hash specification.

97 Higher-order

Author(s): Daniel Cabeza.

This module is a wrapper for the implementation-defined predicate `call/1`, and it implements the `call/2` predicate.

97.1 Usage and interface (`hiord_rt`)

- **Library usage:**
  `:- use_module(library(hiord_rt)).`
- **Exports:**
  - *Predicates:*
    `call/1`, `call/2`, `SYSCALL/1`, `$nodebug_call/1`, `$meta_call/1`.

97.2 Documentation on exports (`hiord_rt`)

call/1: PREDICATE

`call(G)`

Executes goal `G`, restricting the scope of the cuts to the execution of `G`. Equivalent to writing a variable `G` in a goal position.

Meta-predicate with arguments: `call(goal)`.

Usage: ⟨• ISO •⟩

- *The following properties should hold at call time:*
  `G` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

call/2: PREDICATE

`call(Pred,Arg1)`

There exists a set of builtin predicates of the form `call/N` with `N > 1` which execute predicate `Pred` given arguments `Arg1` ... `ArgX`. If `Pred` has already arguments `Arg1` is added to the start, the rest to the end. This predicate, when `Pred` is a variable, can be written using the special Ciao syntax `Pred(Arg1,...,ArgX)`.

Usage:

- *Call and exit should be compatible with:*
  `Arg1` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*
  `Pred` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties should hold upon exit:*
  `Arg1` is any term. (basic_props:term/1)
- *The following properties should hold globally:*
  This predicate is understood natively by CiaoPP. (basic_props:native/1)

SYSCALL/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

$nodebug_call/1: PREDICATE

Meta-predicate with arguments: `$nodebug_call(goal)`.

Usage:

– *The following properties should hold at call time:*

`Arg1` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

$meta_call/1: PREDICATE

Usage: `$meta_call(A)`

– *The following properties should hold at call time:*

`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`A` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties hold globally:*

This predicate is understood natively by CiaoPP as `call(A)`. (basic_props:native/2)

97.3 Known bugs and planned improvements (`hiord_rt`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

98 Higher-order predicates

Author(s): Daniel Cabeza, Manuel Carro, Edison Mera.

This library implements a few basic higher-order predicates. These add functionality to the basic higher-order functionality of Ciao. Examples of the latter are:

Using pred(1):

```
list(L, functor(_,2))
list(L, >(0))
```

Using pred(2):

98.1 Usage and interface (`hiordlib`)

- **Library usage:**
  `:- use_module(library(hiordlib)).`
- **Exports:**
  - *Predicates:*
    `map/3`, `map/4`, `foldl/4`, `minimum/3`, `split/4`.
- **Other modules used:**
  - *System library modules:*
    `assertions/native_props`.

98.2 Documentation on exports (`hiordlib`)

map/3: PREDICATE

Meta-predicate with arguments: `map(?,(pred 2),?)`.

General properties: `map(A,B,C)`

- *If the following properties should hold at call time:*
  term_basic:A=[1,3,2] (term_basic:= /2)
  term_basic:B=arg(f(a,b,c,d)) (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:C=[a,c,b] (term_basic:= /2)
  then the following properties should hold globally:
  All the calls of the form `map(A,B,C)` do not fail. (native_props:not_fails/1)
  All calls of the form `map(A,B,C)` are deterministic. (native_props:is_det/1)

`map(A,B,C)`

- *If the following properties should hold at call time:*
  term_basic:A=[1,3,2] (term_basic:= /2)
  term_basic:B=nth([a,b,c,d]) (term_basic:= /2)
  then the following properties should hold upon exit:
  term_basic:C=[a,c,b] (term_basic:= /2)
  then the following properties should hold globally:
  All the calls of the form `map(A,B,C)` do not fail. (native_props:not_fails/1)
  All calls of the form `map(A,B,C)` are deterministic. (native_props:is_det/1)

`map(A,B,C)`

– *If the following properties should hold at call time:*
term_basic:A=[[68],[67]] (term_basic:= /2)
term_basic:B=append([46]) (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:C=[[68,46],[67,46]] (term_basic:= /2)
then the following properties should hold globally:
All the calls of the form `map(A,B,C)` do not fail. (native_props:not_fails/1)
All calls of the form `map(A,B,C)` are deterministic. (native_props:is_det/1)

Usage: `map(LList,Op,RList)`

– *Description:* Examples of use:

```
map([1,3,2], arg(f(a,b,c,d)), [a,c,b]) or
map([1,3,2], nth([a,b,c,d]), [a,c,b])
map(["D","C"], append("."), ["D.","C."])
```

map/4: PREDICATE

Meta-predicate with arguments: `map(?,(pred 3),?,?)`.

General properties: `map(A,B,C,D)`

– *If the following properties should hold at call time:*
term_basic:A=[1,3,2] (term_basic:= /2)
term_basic:B=((L,[E|T],T):-arg(L,f(a,b,c,d),E)) (term_basic:= /2)
term_basic:D=[x,y] (term_basic:= /2)
then the following properties should hold upon exit:
term_basic:C=[a,c,b,x,y] (term_basic:= /2)
then the following properties should hold globally:
All the calls of the form `map(A,B,C,D)` do not fail. (native_props:not_fails/1)
All calls of the form `map(A,B,C,D)` are deterministic. (native_props:is_det/1)

Usage: `map(LList,Op,RList,Tail)`

– *Description:* DCG version of map.

foldl/4: PREDICATE

Meta-predicate with arguments: `foldl(?,?,(pred 3),?)`.

Usage: `foldl(List,Seed,Op,Result)`

– *Description:* Example of use:

```
?- foldl(["daniel","cabeza","gras"], "",
         (''(X,Y,Z) :- append(X, " "||Y, Z)), R).

R = "daniel cabeza gras " ?
```

minimum/3: PREDICATE

Meta-predicate with arguments: `minimum(?,(pred 2),?)`.

Usage: `minimum(List,SmallerThan,Minimum)`

– *Description:* `Minimum` is the smaller in the nonempty list `List` according to the relation `SmallerThan`: `SmallerThan(X, Y)` succeeds iff X is smaller than Y.

– *The following properties should hold at call time:*

`SmallerThan` is currently a term which is not a free variable. (term_typing:nonvar/1)

`List` is a list. (basic_props:list/1)

`SmallerThan` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Minimum` is any term. (basic_props:term/1)

split/4: PREDICATE

Meta-predicate with arguments: `split(?,(pred 1),?,?)`.

General properties: `split(A,B,C,D)`

– *If the following properties should hold at call time:*

term_basic:A=[1,2,3,4,5,6] (term_basic:= /2)

term_basic:B= >(4) (term_basic:= /2)

then the following properties should hold upon exit:

term_basic:C=[5,6] (term_basic:= /2)

term_basic:D=[1,2,3,4] (term_basic:= /2)

then the following properties should hold globally:

All the calls of the form `split(A,B,C,D)` do not fail. (native_props:not_fails/1)

Usage: `split(List,Condition,Left,Right)`

– *Description:* Divides `List` in two list, where `Left` contains the elements for which the call to `Condition` succeeds, and `Right` the remaining elements.

– *The following properties should hold at call time:*

`List` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Condition` is currently a term which is not a free variable. (term_typing:nonvar/1)

`List` is a list. (basic_props:list/1)

`Condition` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Left` is any term. (basic_props:term/1)

`Right` is any term. (basic_props:term/1)

– *The following properties should hold upon exit:*

`List` is a list. (basic_props:list/1)

`Condition` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Left` is a list. (basic_props:list/1)

`Right` is a list. (basic_props:list/1)

99 Terms with named arguments -records/feature terms

Author(s): Daniel Cabeza, Manuel Hermenegildo, Jose F. Morales.

This library package provides syntax which allows accessing term arguments by name (these terms are sometimes also referred to as *records*, and are also similar to *feature terms* [AKPS92]).

99.1 Usage and interface (argnames_doc)

- **Library usage:**
  `:- use_package(argnames).`
  or
  `:- module(...,...,[argnames]).`
- **Exports:**
  - *Predicates:*
    `$~/3`.
- **New operators defined:**
  `$/2` [150,xfx], `=>/2` [950,xfx], `argnames/1` [1150,fx].
- **New declarations defined:**
  `argnames/1`.

99.2 Documentation on new declarations (argnames_doc)

argnames/1: DECLARATION

Usage: `:- argnames ArgNamedPredSpec.`

– *Description:* An `argnames/1` declaration assigns names to the argument positions of terms (or literal/goals) which use a certain functor/arity. This allows referring to these arguments by their name rather than by their argument position. Sometimes, argument names may be clearer and easier to remember than argument positions, specially for predicates with many arguments. Also, in some cases this may allow adding arguments to certain predicates without having to change the code that uses them. These terms with named arguments are sometimes also referred to as records, and are also similar to feature terms [AKPS92]. For example, in order to write a program for the *zebra* puzzle we might declare:

```
:- use_package([argnames]).
:- argnames house(color, nation, pet, drink, car).
```

which first includes the package and then assigns a name to each of the arguments of any term (or literal/goal) with `house/5` as the main functor.

For convenience the package extends the built-in `data/1` declaration so that names to arguments can be asigned as with the `argnames/1` declaration, as for example:

```
:- data product(id, description, brand, quantity).
```

Once an `argnames/1` is given, it is possible to use the names to refer to the arguments of any term (or literal/goal) which has the same main functor as that of the term which appears in the `argnames/1` declaration. This is done by first writing the functor name, then the infix operator `$`, and then, between curly brackets, zero, one, or more pairs *argument-name*`=>`*argument-value*, separated by commas (i.e., the infix operator

=> is used between the name and the value). Again, argument names must be atomic. Argument values can be any term. Arguments which are not specified are assumed to have a value of "_" (i.e., they are left unconstrained).

Thus, after the declaration for `house/5` in the example above, any ocurrence in that code of, for example, `house${nation=>Owns_zebra,pet=>zebra}` is exactly equivalent to `house(_,Owns_zebra,zebra,_,_)`. Also, `house${}` is equivalent to `house(_,_,_,_,_)`. The actual zebra puzzle specification might include a clause such as:

```
zebra(Owns_zebra, Drinks_water, Street) :-
   Street = [house${},house${},house${},house${},house${}],
   member(house${nation=>Owns_zebra,pet=>zebra}, Street),
   member(house${nation=>Drinks_water,drink=>water}, Street),
   member(house${drink=>coffee,color=>green}, Street),
   left_right(house${color=>ivory}, house${color=>green}, Street),
   member(house${car=>porsche,pet=>snails}, Street),
        ...
```

Another syntax supported, useful mainly in declarations to avoid specifying the arity, is `house${/}`, which is equivalent in our example to `house/5` (but for data declarations there is a special syntax as we have seen).

Any number of `argnames/1` declarations can appear in a file, one for each functor whose arguments are to be accessed by name. As with other packages, argument name declarations are *local to the file* in which they appear. The `argnames/1` declarations affect only program text which appears after the declaration. It is easy to make a set of declarations affect several files for example by putting such declarations in a sepatate file which is included by all such files.

An `argnames/1` declaration does not change in any way the internal representation of the associated terms and does not affect run-time efficiency. It is simply syntactic sugar.

Runtime support

It is possible to write pairs with unbound argument names. In that case, runtime information is emitted to resolve the argument name at execution time.

99.3 Documentation on exports (argnames_doc)

$~/3: PREDICATE

Usage: `$~(Term,Replacement,NewTerm)`

– *Description:* `NewTerm` is as `Term` but with the arguments specified in `Replacement` changed (they need to be in argnames syntax). The predicate is in fact virtual, since it is translated by the package to a pair of unifications. For example, given the declaration `:- argnames house(color, nation, pet, drink, car)`, the goal

```
$~(House, house${car => seat, pet => mouse}, NewHouse)
```

would be compiled to the unifications

```
House = house(C,N,_,D,_), NewHouse = house(C,N,mouse,D,seat).
```

99.4 Other information (argnames_doc)

Two simple examples of the use of the argnames library package follow.

99.4.1 Using argument names in a toy database

```
:- module(simple_db,_,[argnames,assertions,regtypes]).
:- use_module(library(aggregates)).

:- doc(title,"A simple database application using argument names").

:- data
product( id,    description,    brand,          quantity        ).
%       ------------------------------------------------------
product(  1,    "Keyboard",     "Logitech",     6               ).
product(  2,    "Mouse",        "Logitech",     5               ).
product(  3,    "Monitor",      "Philips",      3               ).
product(  4,    "Laptop",       "Dell",         4               ).
% (${/} must go after argnames)
:- pred product${/}
   ::    int    * string        * string        * int.

% Compute the stock of products from a given brand.
% Note call to findall is equivalent to: findall(Q,product(_,_,Brand,Q),L).

brand_stock(Brand,Stock) :-
        findall(Q,product${brand=>Brand,quantity=>Q},L),
        sumlist(L,Stock).

sumlist([],0).
sumlist([X|T],S) :-
        sumlist(T,S1),
        S is X + S1.
```

99.4.2 Complete code for the zebra example

```
:- module(_,zebra/3,[argnames]).

/*     There are five consecutive houses, each of a different
color and inhabited by men of different nationalities. They each
own a different pet, have a different favorite drink, and drive a
different car.

1.   The Englishman lives in the red house.
2.   The Spaniard owns the dog.
3.   Coffee is drunk in the green house.
4.   The Ukrainian drinks tea.
5.   The green house is immediately to the right of the ivory
     house.
6.   The Porsche driver owns snails.
7.   The Masserati is driven by the man who lives in the yellow
     house.
8.   Milk is drunk in the middle house.
9.   The Norwegian lives in the first house on the left.
```

```
10.  The man who drives a Saab lives in the house next to the man
     with the fox.
11.  The Masserati is driven by the man in the house next to the
     house where the horse is kept.
12.  The Honda driver drinks orange juice.
13.  The Japanese drives a Jaguar.
14.  The Norwegian lives next to the blue house.

The problem is: Who owns the Zebra?  Who drinks water?
*/

:- argnames house(color, nation, pet, drink, car).

zebra(Owns_zebra, Drinks_water, Street) :-
        Street = [house${},house${},house${},house${},house${}],
        member(house${nation => Owns_zebra, pet => zebra}, Street),
        member(house${nation => Drinks_water, drink => water}, Street),
        member(house${nation => englishman, color => red}, Street),
        member(house${nation => spaniard, pet => dog}, Street),
        member(house${drink => coffee, color => green}, Street),
        member(house${nation => ukrainian, drink => tea}, Street),
        left_right(house${color => ivory}, house${color => green}, Street),
        member(house${car => porsche, pet => snails}, Street),
        member(house${car => masserati, color => yellow}, Street),
        Street = [_, _, house${drink => milk}, _, _],
        Street = [house${nation => norwegian}|_],
        next_to(house${car => saab}, house${pet => fox}, Street),
        next_to(house${car => masserati}, house${pet => horse}, Street),
        member(house${car => honda, drink => orange_juice}, Street),
        member(house${nation => japanese, car => jaguar}, Street),
        next_to(house${nation => norwegian}, house${color => blue}, Street).


member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).

left_right(L,R,[L,R|_]).
left_right(L,R,[_|T]) :- left_right(L,R,T).

next_to(X,Y,L) :- left_right(X,Y,L).
next_to(X,Y,L) :- left_right(Y,X,L).
```

99.5 Known bugs and planned improvements (argnames_doc)

- It would be nice to add a mechanism to portray terms with named arguments in a special (user definable) way.
- The predicate `$~` still does not support runtime argnames.

100 Functional notation

Author(s): Daniel Cabeza, Amadeo Casas, Manuel Hermenegildo, Jose F. Morales.

This library package allows the use of functional notation in a Ciao module/program. It supports function application, predefined evaluable functors, functional definitions, quoting, and (combined with the `lazy` library) lazy evaluation. The extensions implemented by this library are also composable with higher-order features and can be combined with other Ciao packages such as constraints, assertions, etc.

The package provides *syntactic sugar* for defining and using predicates as if they were functions. However, they can still retain the power of predicates. Any function definition written using this package is in fact defining a predicate, and any predicate can be used as a function.

The predicate associated with a function has the same name and one more argument, meant as the place holder for the "result" of the function. In fact, this argument is just the one that will be syntactically connected to the surrounding goal or function, but it does not necessarily imply any directionality, i.e., it does not necessarily mean that this argument is an output or an input. This argument is by default added to the right, i.e., it is the last argument, but can be changed by using a declaration, as explained below.

100.1 Function applications

Any term preceded by the `~ /1` operator is a function application, as can be seen in the goal `write(~arg(1,T))`, which is strictly equivalent to the sequence `arg(1,T,A), write(A)`. The declaration `fun_return/1` allows using a predicate argument other than the last as the return argument. For example with `:- fun_return functor(~,_,_)` the expression `~functor(f,2)` will be evaluated to the term `f(_,_)`. This definition of the return argument can also be done on the fly in each invocation in the following way: `~functor(~,f,2)`.

Functors can be declared as evaluable by using the declaration `fun_eval/1`. This allows avoiding the need to use the `~` operator. Thus, `:- fun_eval arg/2` allows writing `write(arg(1,T))` instead of `write(~arg(1,T))` as above. This declaration can be combined with the previous one: `:- fun_eval functor(~,_,_)`.

100.2 Predefined evaluable functors

By using the declaration `:- fun_eval arith(true)`, all the functors understood by `is/2` will be also evaluated. This is active from the declaration downwards until a `:- fun_eval arith(false)` declaration or the end of the module is reached. Beware that arithmetic functors are used in some cases for other purposes than arithmetic: e.g. `abolish(p/2)`. But this is not so disturbing as it may appear because this package is not active in declarations, except for the goal-including declarations `initialization/1` and `on_abort/1`. Note that all the declarations introduced by this package, as is customary in Ciao, are local to the module where they are included.

In addition to functors declared with the declaration `fun_eval/1`, the package defines as evaluable the functors used for disjunctive and conditional expressions: `| /2` and `? /2` (defined as operators). A disjunctive expression has the form `(V1|V2)`, and its value when first evaluated is `V1`, and on backtracking `V2`. A conditional expression has the form `(Cond ? V1)`, or more commonly `(Cond ? V1 | V2)`, and its value, if the execution of `Cond` as a goal succeeds, is `V1`, otherwise in the first form it causes backtracking, and on the second form its value is `V2`. Note that due to the operator precedences, these expressions normally need to be surrounded by parenthesis. Also, a nested expression: `(Cond1 ? V1 | Cond2 ? V2 | V3)` is evaluated as `(Cond1 ? V1 | (Cond2 ? V2 | V3))`.

100.3 Functional definitions

A functional definition is composed of one or more functional clauses. A functional clause is written using the binary operator `:= /2`, as in:

```
opposite(red) := green.
```

which is equivalent to `opposite(red,green).` or

```
addlast(X,L) := ~append(L,[X]).
```

which is equivalent to `addlast(X,L,R) :- append(L,[X],R).`

Functional clauses can also have a body, which is executed before the result value is computed. It can serve as a guard for the clause or to provide the equivalent of where-clauses in functional languages:

```
fact(0) := 1.
fact(N) := N * ~fact(--N) :- N > 0.
```

Note that guards can often be defined more compactly using conditional expressions:

```
fact(N) := N = 0 ? 1
         | N > 0 ? N * ~fact(--N).
```

The declaration `:- fun_eval defined(true)` allows to locally define as evaluable functions being defined, so that the `~` operator does not need to be used within a functional definition for the functor being defined. For example, for the `fact` invocations in the previous definitions, which can now be written as, e.g. (we provide the full module definition):

```
:- module(_,_,[fsyntax]).

:- fun_eval arith(true).
:- fun_eval defined(true).

fact(0) := 1.
fact(N) := N * fact(--N) :- N > 0.

%% Or,alternatively:
%
% fact(N) := N=0 ? 1
%          | N>0 ? N * fact(--N).
```

This behaviour is reverted using `:- fun_eval defined(false)`.

The translation of functional clauses has the following properties:

- The translation produces *steadfast* predicates, that is, output arguments are unified after possible cuts.
- Defining recursive predicates in functional style maintains the tail recursion of the original predicate, thus allowing the usual compiler optimizations.

Some implementation details and a discussion of the recent combination of this library (which dates from Ciao version 0.2) with the lazy evaluation library can be found in [CCH06].

100.4 Quoting functors

Functors (either in functional or predicate clauses) can be prevented from being evaluated by using the `^ /1` prefix operator (read as "quote"), as in

```
:- fun_eval arith(true).
pair(A,B) := ^(A-B).
```

Note that this just prevents the evaluation of the principal functor of the enclosed term, not the possible occurrences of other evaluable functors inside.

100.5 Some scoping issues

When using function applications inside the goal arguments of meta-predicates, there is an ambiguity as they could be evaluated either in the scope of the outer execution or the in the scope of the inner execution. The chosen behavior is by default to evaluate function applications in the scope of the outer execution. If they should be evaluated in the inner scope, the goal containing the function application needs to be escaped with the `^^ /1` prefix operator, as in `findall(X, (d(Y), ^^(X = ~f(Y)+1)), L)` (which could also be written as `findall(X, ^^(d(Y), X = ~f(Y)+1), L)`) and which expands into `findall(X, (d(Y),f(Y,Z),T is Z+1,X=T), L)`. With no escaping the function application is evaluated in the scope of the outer execution, i.e., it expands to `f(Y,Z), T is Z+1, findall(X, (d(Y),X=T), L)`.

100.6 Other functionality

In addition to the basic package `fsyntax`, a package `functional` is also provided, to allow programming with a more functional-flavored style. That package activates the declarations `:- fun_eval arith(true)` and `:- fun_eval defined(true)`, and defines the `. /2` operator for use in lists (but be careful: this period cannot be followed by a whitespace!) and the operator `++ /2` as a function for appending lists. The factorial example above can be written as follows using the `functional` package:

```
:- module(_,_,[fsyntax]).

:- fun_eval arith(true).
:- fun_eval defined(true).

fact(0) := 1.
fact(N) := N * fact(--N) :- N > 0.

%% Or,alternatively:
%
% fact(N) := N=0 ? 1
%          | N>0 ? N * fact(--N).
```

See the end of this chapter for additional examples.

100.7 Combining with higher order

Ciao provides in its standard library the `hiord` package, which supports a form of higher-order untyped logic programming with predicate abstractions [CH99a,Cab04,CHL04]. Predicate abstractions are Ciao's translation to logic programming of the lambda expressions of functional programming: they define unnamed predicates which will be ultimately executed by a higher-order call, unifying its arguments appropriately. A function abstraction is provided as functional syntactic sugar for predicate abstractions:

Predicate abstraction: `''(X,Y) :- p(X,Z), q(Z,Y).`

Function abstraction: `''(X) := ~q(~p(X)).`

and function application is syntactic sugar over predicate application:

Predicate application: `..., P(X,Y), ...` Function application: `..., Y = ~P(X), ...`

The combination of this `hiord` package with the `fsyntax` and `lazy` packages (and, optionally, the type inference and checking provided by the Ciao preprocessor [HPBLG05]) basically provide the functionality present in modern functional languages (currying is not *syntactically* implemented, but its results can be obtained by deriving higher-order data from any other higher-order data (see [Cab04]), as well as some of the functionality of full higher-order logic programming.

At this moment, it is necessary to specify the `:- fun_eval hiord(true)` option to enable correct handling of function abstractions.

100.8 Usage and interface (`fsyntax_doc`)

- **Library usage:**

  `:- use_package(fsyntax).`

  or

  `:- module(...,...,[fsyntax]).`

100.9 Other information (`fsyntax_doc`)

100.10 Some examples using functional syntax

We now illustrate some of the uses of the package through examples. The following example defines a simple unary function `der(X)` which returns the derivative of a polynomial arithmetic expression:

```
der(x)      := 1.
der(C)      := 0                  :- number(C).
der(A + B)  := der(A) + der(B).
der(C * A)  := C * der(A)         :- number(C).
der(x ** N) := N * x ** ~(N - 1) :- integer(N), N > 0.
```

Note that if we include the directive mentioned before which makes arithmetic functors evaluable then we would have to write the program in the following (clearly, less pleasant and more obfuscated) way:

```
:- fun_eval(arith(true)).
der(x)         := 1.
der(C)         := 0                        :- number(C).
der(^(A + B))  := ^(der(A) + der(B)).
der(^(C * A))  := ^(C * der(A))            :- number(C).
der(^(x ** N)) := ^(N * ^(x ** (N - 1))) :- integer(N), N > 0.
```

Both of the previous code fragments translate to the following code:

```
der(x, 1).
der(C, 0) :-
          number(C).
der(A + B, X + Y) :-
          der(A, X),
```

```
          der(B, Y).
der(C * A, C * X) :-
          number(C),
          der(A, X).
der(x ** N, N * x ** N1) :-
          integer(N),
          N > 0,
          N1 is N - 1.
```

Functional notation interacts well with other Ciao language features. For example, it provides compact and familiar notation for regular types and other properties:

```
:- module(_,_,[hiord,functional,assertions,regtypes,'bf/bfall']).

:- regtype color/1. color := red | blue | green.

:- regtype slist/1. slist := [] | [ _ | slist].

:- regtype list_of/1. list_of(T) := [] | [~T | list_of(T)].
```

where the functional clauses expand to (note the use of higher-order in the third example):

```
color(red). color(blue). color(green).
list([]).
list([_|T]) :- list(T).
list_of(_, []).
list_of(T, [X|Xs]) :- T(X), list_of(T, Xs).
```

Such types and properties are then admissible in the usual way in assertions, e.g.:

```
:- pred append/3 :: list * list * list.
:- pred color_value/2 :: list(color) * int.
```

The combination of functional syntax and user-defined operators brings significant flexibility, as can be seen in the following definition of a list concatenation (`append`) operator (note that these are the definitions mentioned before which are active by default in the `functional` package):

```
:- op(600, xfy, (.)).
:- op(650, xfy, (++)).
:- fun_eval (++)/2.
[]   ++ L := L.
X.Xs ++ L := X.(Xs ++ L).
```

This definition will be compiled exactly to the standard definition of `append` (and, thus, will be reversible). The functional syntax and user-defined operators allow writing for example `Space = ' ', write("Hello" ++ Space ++ "world!")` instead of the equivalent forms `Space = ' ', write( append("Hello", append(Space, "world!")))` (if `append/2` is defined as evaluable) or `Space = ' ', append(Space, "world!", T1), append("Hello", T1, T2), write(T2)`.

As another example, we can define an array indexing operator for fixed-size, multi-dimensional arrays. Assume that arrays are built using nested structures whose main functor is `a` and whose arities are determined by the specified dimensions, i.e., a two-dimensional array *A* of dimensions *[N,M]* will be represented by the nested structure `a(a(`*A11*`,...,`*A1M*`), a(`*A21*`,..,`*A2M*`), ..., a(`*AN1*`,..., `*ANM*`))`, where *A11*,... *ANM* may be arbitrary terms (we ignore for simplicity arity limitations, solved in any case typically by further nesting with logarithmic access time). The following recursive definition defines the property `fixed_array/2` and also the array access operator `@`:

```
fixed_array([N|Ms],A):-
```

```
functor(A,a,N),
rows(N,Ms,A).
fixed_array([N],A):-
functor(A,a,N).

rows(0,_,_).
rows(N,Ms,A) :-
        N > 0,
        arg(N,A,Arg),
        array(Ms,Arg),
        rows(N-1,Ms,A).

:- pred @(Array,Index,Elem) :: array * list(int) * int
   # "@var{Elem} is the @var{Index}-th element of @var{Array}.".

:- op(55, xfx, '@').
:- fun_eval (@)/2.
V@[I]    := ~arg(I,V).       %% Or: V@[] := V.
V@[I|Js] := ~arg(I,V)@Js.
```

This allows writing, e.g., `M = fixed_array([2,2]), M@[2,1] = 3` (which could also be expressed as `fixed_array([2,2])@[2,1] = 3`), where the call to the `fixed_array` property generates an empty *2 x 2* array *M* and `M@[2,1] = 3` puts *3* in *M[2,1]*. This can be done in the top level:

```
?- M = ~fixed_array([2,2]), M@[2,1] = 3.
```

provided the `op` and `function` declarations are loaded into the top level also. Another example of use is: `A3@[N+1,M] = A1@[N-1,M] + A2@[N,M+2]`.

Such functionality can be grouped into a *package* as follows. The package main file (`arrays.pl`) might be:

```
:- package(arrays).
:- include(arrays_ops).

:- use_module(arrays_rt).
```

where file `arrays_ops.pl` may contain:

```
:- use_package(functional).

:- op(150,xfx,[@]).
:- fun_eval '@'/2.

:- op(500,yfx,<+>).
:- fun_eval '<+>'/2.

:- op(400,yfx,<*>).
:- fun_eval '<*>'/2.
```

The main file is `arrays_rt.pl` which would contain for example (note that it also uses `arrays_ops.pl`, and that is why the contents of `arrays_ops.pl` were not put directly in `arrays.pl`):

```
:- module(arrays_rt,_,[functional,hiord,assertions,regtypes,isomodes]).
```

```
:- include(arrays_ops).

:- doc(title,"Some simple array operations with syntactic support").
:- doc(author,"Pro Grammer").

:- doc(module,"This library implements a very simple set of
   operations on arrays. The idea is to illustrate the use of
   functional syntax (operators) by providing syntactic support for
   invoking array operations such as element access, array (vector)
   addition, etc.").

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Regtypes

%% :- doc(doinclude,array/1).
%% :- doc(doinclude,vector/1).
%% :- doc(doinclude,dim/1).

:- regtype array(A) #"@var{A} is a multi-dimensional array.".
% Should obviously be defined in more detail...
array(A) :- struct(A).

:- regtype dim(D) # "@var{D} represents the dimensions of an array.".
dim(D) :- list(D,int).

:- regtype vector(V) # "@var{V} is a one-dimensional fixed-size array.".
vector(V) :- fixed_array([N],V), int(N).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- pred fixed_array(Dim,Array) :: dim * array
# "@var{Array} is an array of fixed dimensions @var{Dim}.".

fixed_array([N|Ms],A):-
        functor(A,a,N),
        rows(N,Ms,A).
fixed_array([N],A):-
        functor(A,a,N).

rows(0,_Ms,_A).
rows(N,Ms,A):-
        N > 0,
        arg(N,A,Arg),
        fixed_array(Ms,Arg),
        rows(N-1,Ms,A).

:- pred @(Array,Index,Elem):: array * dim * int
# "@var{Elem} is the @var{Index}-th element of @var{Array}.".

V@[I]    := ~arg(I,V).
V@[I|Js] := ~arg(I,V)@Js.
```

```
:- pred <+>(V1,V2,V3) :: vector * vector * vector
# "@var{V3} is @var{V1} + @var{V2}.".

V1 <+> V2 := V3 :-
        V1 = ~fixed_array([N]),
        V2 = ~fixed_array([N]),
        V3 = ~fixed_array([N]),
        V3 = ~vecplus_(N,V1,V2).

vecplus_(0,_,_,_).
vecplus_(N,V1,V2,V3) :-
        N > 0,
        V3@[N] = V1@[N] + V2@[N],
        vecplus_(N-1,V1,V2,V3).

:- pred <*>(V1,V2,V3) :: vector * vector * vector
# "@var{V3} is @var{V1} * @var{V2} (inner product).".

V1 <*> V2 := ~vecmul_(N,V1,V2,0) :-
    V1 = ~fixed_array([N]),
    V2 = ~fixed_array([N]).

vecmul_(0,  _,  _, Acc, Acc).
vecmul_(N, V1, V2, Acc, IP) :-
    N > 0,
    vecmul_( N-1, V1, V2, Acc + ( V1@[N] * V2@[N] ), IP).
```

A file using this package would be:

```
:- module(_,_).

:- use_package(library(fsyntax(examples(arrays)))).

main(M) :-
        V1 = a(1,3,4,5),
        V2 = a(5,4,3,1),
        I = 1,
        display(V2@[I+1]),
        M = V1 <*> ( V2 <+> V1 ).
```

100.11 Examples of combining with higher order

The following `map` and `foldl` definitions (from the `hiordlib` library) illustrate the combination of functional syntax and higher-order logic programming:

```
:- fun_eval map/2.
:- meta_predicate map(_,pred(2),_).
map([], _)     := [].
map([X|Xs], P) := [P(X) | map(Xs, P)].

:- fun_eval foldl/3.
```

```
:- meta_predicate foldl(_,_,pred(3),_).
foldl([], Seed, _Op) := Seed.
foldl([X|Xs], Seed, Op) := ~Op(X,~foldl(Xs,Seed,Op)).
```

With this definition:

```
?- L = ~map([1,2,3], ( _(X,Y):- Y = f(X) ) ).

L = [f(1),f(2),f(3)] ?

?- [f(1),f(2),f(3)] = ~map(L, ( _(X,f(X)) :- true ) ).

L = [1,2,3] ?
```

Also, after running:

```
?- ["helloworld", "byeworld"] = map(["hello", "bye"], ++(X)).
```

(where `(++)/2` corresponds to the above definition of `append`) `X` will be bound to `"world"`, which is the only solution to the equation.

And when calling:

```
map(L, ++(X), ["hello.", "bye."]).
```

several values for `L` and `X` are returned through backtracking:

```
L = ["hello","bye"],   X = "." ? ;
L = ["hello.","bye."], X = [] ?
```

(remember to set the flag `write_strings` to on in these examples so that the top level prints strings as strings of characters instead of lists of ASCII codes).

100.12 Some additional examples using functional syntax

A definition of the Fibonacci function, written in functional notation:

```
:- module(_,_,[functional]).

fib(0) := 0.
fib(1) := 1.
fib(N) := fib(N-1) + fib(N-2) :- integer(N), N > 1.

write_fib(N):-
        message(['The ',N,'. Fibonacci number is: ',~fib(N),'.']).
```

This is the factorial example, written in functional notation and including some assertions:

```
:- module(_,_,[assertions,nativeprops,functional]).

:- pred fact(+int,-int) + is_det.
:- pred fact(-int,+int) + non_det.

fact(N) := N=0 ? 1
         | N>0 ? N * fact(--N).
```

And, the same example written using `clpq` constraints:

```
:- module(_,_,[assertions,nativeprops,fsyntax,clpqf]).

:- fun_eval .=. /1.
```

```
:- op(700,fx,[.=.]).
:- fun_eval fact/1.

:- pred fact(+int,-int) + is_det.
:- pred fact(-int,-int) + non_det.

fact( .=. 0) := .=. 1.
fact(N) := .=. N*fact( .=. N-1 ) :- N .>. 0.
```

which allows for example calling it "backwards:"

```
?- 24 = ~fact(X).

X = 4 ?
```

A very simple example using lazy evaluation:

```
:- module(_,_,[functional,lazy]).
:- use_module(library(lazy(lazy_lib)), [take/3]).

nums(N) := ~take(N,nums_from(0)).

:- lazy fun_eval nums_from/1.

nums_from(X) := [X | nums_from(X+1)].
```

A naive reverse example, using functional notation:

```
:- module(_, [nrev/2], [functional]).

nrev( [] )    := [].
nrev( [H|T] ) := ~conc( nrev(T),[H] ).

conc( [],    L ) := L.
conc( [H|T], K ) := [ H | conc(T,K) ].
```

And the same example using some assertions:

```
:- module(_, [nrev/2], [assertions,fsyntax,nativeprops]).

:- entry nrev/2 : {list, ground} * var.

:- pred nrev(A,B) : list(A) => list(B)
   + ( not_fails, is_det, steps_o( exp(length(A),2) ) ).

nrev( [] )    := [].
nrev( [H|L] ) := ~conc( ~nrev(L),[H] ).


:- pred conc(A,_,_) + ( terminates, is_det, steps_o(length(A)) ).

conc( [],    L ) := L.
conc( [H|L], K ) := [ H | ~conc(L,K) ].
```

Finally, a simple stream creation example where assertions are used to define a safety policy (that no file outside `/tmp` should be opened):

```
:- module(_,[create_streams/2],[fsyntax,assertions,regtypes]).

:- entry create_streams(A,B) : list(A,num).

create_streams([])     := [].
create_streams([N|NL]) := [ ~open_file(Fname,write) | ~create_streams(NL) ]
        :-
         app("/tmp/../",~number_codes(N),Fname).
%        app("/tmp/",~number_codes(N),Fname).

app([],L) := L.
app([X|Xs],L) := [X|~app(Xs,L)].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% open_file library:

open_file(Fname,Mode) := ~open(File,Mode) :- atom_codes(File,Fname).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Safety policy:

:- check calls open_file(Fname,_,_) : safe_name(Fname).

:- regtype  safe_name/1.    safe_name("/tmp/" || L) :- list(L,alphnum_code).

:- regtype  alphnum_code/1. alphnum_code := ~alph_code | ~num_code.

:- regtype  alph_code/1.    alph_code := 0'a | 0'b | 0'c | 0'd | 0'e | 0'f .

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

100.13 Known bugs and planned improvements (`fsyntax_doc`)

- Assumes that `is/2` is imported.
- Lazy functions declarations require translation priorities to move it to the lazy package.
- Detect automatically when hiord is being used, deprecate `eval_hiord`.
- I am not sure if shared variables are working for predicate abstractions.
- Find out if predicate abstractions are being fully translated at compile time (see output for hiordfun example).

101 global (library)

101.1 Usage and interface (global)

- **Library usage:**
  `:- use_module(library(global)).`
- **Exports:**
  - *Predicates:*
    `set_global/2`, `get_global/2`, `push_global/2`, `pop_global/2`, `del_global/1`.

101.2 Documentation on exports (global)

set_global/2: PREDICATE
No further documentation available for this predicate.

get_global/2: PREDICATE
No further documentation available for this predicate.

push_global/2: PREDICATE
No further documentation available for this predicate.

pop_global/2: PREDICATE
No further documentation available for this predicate.

del_global/1: PREDICATE
No further documentation available for this predicate.

102 Andorra execution

Author(s): Claudio Vaucheret, Francisco Bueno.

This package allows the execution under the Basic Andorra Model [War88]. The model classifies goals as a *determinate goal*, if at most one clause matches the goal, or nondeterminate goal, otherwise. In this model a goal is delayed until either it becomes determinate or it becomes the leftmost goal and no determinate goal is available. The implementation of this selection rule is based on the use of attributed variables [Hol92,Hol90].

In order to test determinacy we verify only the heads of clauses and builtins in the bodies of clauses before the first cut, if any. By default, determinacy of a goal is detected dynamically: when called, if at most one clause matches, it is executed; otherwise, it is delayed. For goals delayed the test is repeated each time a variable appearing in the goal is instantiated. In addition, efficiency can be improved by using declarations that specify the determinacy conditions. These will be considered for testing instead of the generic test on all clauses that can match.

As with any other Ciao package, the andorra computation rule affects only the module that uses the package. If execution passes across two modules that use the computation rule, determinate goals are run in advance *within* one module and also within the other module. But determinate goals of one module do not run ahead of goals of the other module.

It is however possible to preserve the computation rule for calls to predicates defined in other modules. These modules should obviously also use this package. In addition *all* predicates from such modules should imported, i.e., the directive `:- use_module(module)`, should be used in this case instead of `:- use_module(module,[...])`. Otherwise calls to predicates outside the module will only be called when they became the leftmost goal.

102.1 Usage and interface (andorra_doc)

- **Library usage:**
  `:- use_package(andorra).`
  or
  `:- module(...,...,[andorra]).`
- **Exports:**
  - *Regular Types:*
    `detcond/1`, `path/1`.
- **New operators defined:**
  `?\=/2` [700,xfx], `?=/2` [700,xfx].
- **New declarations defined:**
  `determinate/2`.

102.2 Documentation on new declarations (andorra_doc)

determinate/2: DECLARATION

`:- determinate(Pred,Cond).`

Declares determinacy conditions for a predicate. Conditions `Cond` are on variables of arguments of `Pred`. For example, in:

```
:- determinate(member(A,B,C), ( A ?= term(B,[1])  ; C?=[_|_]) ).

member(A,[A|B],B).
member(A,[B|C],[B|D]) :-
        A==B,
        member(A,C,D).
```

the declaration states that a call `member(A,B,C)` is determinate when either `A` doesn't unify with the first argument of `B` or `C` doesn't unify with `[_|_]`.

Usage: :- `determinate(Pred,Cond)`.

- *Description:* States that the predicate `Pred` is determinate when `Cond` holds.
- *The following properties should hold at call time:*

  `Pred` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

`Cond` is a determinacy condition. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/library/andorra/andorra_doc):detcond/1)

102.3 Documentation on exports (andorra_doc)

detcond/1: REGTYPE

Defined by:

```
detcond(ground(X)) :-
        var(X).
detcond(nonvar(X)) :-
        var(X).
detcond(instatiated(A,Path)) :-
        var(A),
        list(Path,int).
detcond(?\=(Term1,Term2)) :-
        path(Term1),
        path(Term2).
detcond(?=(Term1,Term2)) :-
        path(Term1),
        path(Term2).
detcond(Test) :-
        ( test Test ).
```

- `ground/1` and `nonvar/1` have the usual meaning.
- `instatiated(A,Path)` means that the subterm of `A` addressed by `Path` is not a variable. `Path` is a list of integer numbers describing a path to the subterm regarding the whole term A as a tree. For example, `instantiated(f(g(X),h(i(Z),Y)),[2,1])` tests whether `i(Z)` is not a variable.
- `Term1 ?\= Term2` means "terms `Term1` and `Term2` do not unify (when instantiated)". `Term1` and `Term2` can be either an argument of the predicate or a term `term(V,Path)`, which refers to the subterm of `V` addressed by `Path`.
- `Term1 ?= Term2` means "terms `Term1` and `Term2` unify (when instantiated)". The same considerations above apply to `Term1` and `Term2`.

- any other test that does not unify variables can also be used (`==/2`, `\==/2`, `atomic/1`).

Usage: `detcond(X)`

– *Description:* `X` is a determinacy condition.

path/1: REGTYPE

Defined by:

```
path(X) :-
        var(X).
path(X) :-
        list(X,int).
```

102.4 Other information (`andorra_doc`)

The andorra transformation will include the following predicates into the code of the module that uses the package. Be careful not to define predicates by these names:

- `detcond_andorra/4`
- `path_andorra/4`
- `detcond_susp/4`
- `path_susp/4`
- `list_andorra2/5`
- `test_andorra2/4`

103 And-parallel execution

Author(s): Amadeo Casas (`http://www.cs.unm.edu/~amadeo`, University of New Mexico).

This library allows and-parallel execution of goals in (Herbrand-)independent fashion. It resembles the execution rules of &-Prolog.

103.1 Usage and interface (`andprolog_doc`)

- **Library usage:**
  The AND_PARALLEL_EXECUTION flag must be set to "yes" in order to compile the engine with support for the and-parallel execution of goals. Concurrency primitives are defined in the `apll` library. That flag may be set to "visandor" to, in addition, add support for VisAndOr's events for deterministic parallel goals.
- **New operators defined:**
  `&/2` [950,xfy], `&>/2` [950,xfy], `&&>/2` [950,xfy], `&&/2` [950,xfy], `<&/1` [950,xf], `<&&/1` [950,xf], `&/1` [950,xf], `&&/1` [950,xf], `&!/2` [950,xfy], `&!>/2` [950,xfy], `&&!>/2` [950,xfy], `&&!/2` [950,xfy], `<&!/1` [950,xf], `<&&!/1` [950,xf], `&!/1` [950,xf], `&&!/1` [950,xf], `<?/1` [950,xf], `&?/1` [950,xf].
- **Other modules used:**
  - *System library modules:*
    `andprolog/andprolog_rt`.

104 Low-level concurrency primitives for and-parallelism support

Author(s): Amadeo Casas (`http://www.cs.unm.edu/~amadeo`, University of New Mexico).

This module provides basic mechanisms to start threads, wait for their completion, push goals, search for goals, access to locks, etc. Most of these primitives need to refer to an explicit goal and need to use some information related to its state, stored in the data structure `Handler`.

This primitives allow to efficiently implement at a higher-level different approaches to exploiting independent and-parallelism.

104.1 Usage and interface (apll)

- **Library usage:**

  `:- use_package(apll).`

  or

  `:- module(...,...,[apll]).`
- **Exports:**
  - *Predicates:*

    initial/0, $start_thread/1, $number_agents/1, $push_goal/3, $find_goal/3, $goal_available/1, $cancellation/1, $retrieve_goal/2, $goal_det/1, $set_goal_det/1, $set_goal_nondet/1, $goal_not_executed/1, $set_goal_not_executed/1, $goal_rem_executing/1, $set_goal_rem_executing/1, $goal_finished/1, $set_goal_finished/1, $goal_tobacktrack/1, $set_goal_tobacktrack/1, $goal_toreexecute/1, $set_goal_toreexecute/1, $goal_failed/1, $set_goal_failed/1, $goal_cancelled/1, $show_handler/1, $set_goal_cancelled/1, $send_event/1, $read_event/1, $save_init_execution/1, $save_end_execution/1, $more_solutions/1, $move_execution_top/1, $waiting/1, $suspend/0, $release/1, $release_remote/1, $release_some_suspended_thread/0, $release_all_for_unwinding/0, $enter_mutex/1, $enter_mutex_self/0, $enter_mutex_remote/1, $exit_mutex/1, $exit_mutex_self/0, $exit_mutex_remote/1, $clean_measures/0, $print_measures/0, $new_measure/0, $not_measure/0, $incr_num_local_backtr/0, start_thread/1, number_agents/1, push_goal/3, push_goal/1, find_goal/2, find_det_goal/2, goal_available/1, cancellation/1, retrieve_goal/2, goal_det/1, set_goal_det/1, set_goal_nondet/1, goal_not_executed/1, set_goal_not_executed/1, goal_rem_executing/1, set_goal_rem_executing/1, goal_finished/1, set_goal_finished/1, goal_tobacktrack/1, set_goal_tobacktrack/1, goal_toreexecute/1, set_goal_toreexecute/1, goal_failed/1, set_goal_failed/1, show_handler/1, goal_cancelled/1, set_goal_cancelled/1, send_event/1, read_event/1, save_init_execution/1, save_end_execution/1, more_solutions/1, move_execution_top/1, waiting/1, suspend/0, release/1, release_remote/1, release_some_suspended_thread/0, release_all_for_unwinding/0, enter_mutex/1, enter_mutex_self/0, enter_mutex_remote/1, exit_mutex/1, exit_mutex_self/0, exit_mutex_remote/1, clean_measures/0, print_measures/0, new_measure/0, not_measure/0, incr_num_local_backtr/0.
- **New operators defined:**

  `?/1` [200,fy], `@/1` [200,fy], `?/1` [500,fx], `@/1` [500,fx], `regtype/1` [1150,fx], `regtype/2` [1150,xfx], `-->/2` [1100,xfy].
- **New modes defined:**

  `+/1`, `@/1`, `-/1`, `?/1`, `*/1`, `+/2`, `@/2`, `-/2`, `?/2`, `*/2`, `in/1`, `out/1`, `go/1`, `in/2`, `out/2`, `go/2`.
- **New declarations defined:**

  `regtype/1`, `regtype/2`.
- **Other modules used:**
  - *System library modules:*

    foreign_interface/foreign_interface_properties, apll/visandor.

104.2 Documentation on new declarations (`apll`)

regtype/1: (UNDOC_REEXPORT)
Imported from `basic_props` (see the corresponding documentation for details).

regtype/2: DECLARATION
No further documentation available for this predicate.

104.3 Documentation on new modes (`apll`)

+/1: MODE

Usage 1: `+A`

– *The following properties are added at call time:*
`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 2: `+A`

– *The following properties are added at call time:*
`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

@/1: MODE

Usage 1: `@A`

– *The following properties are added globally:*
`A` is not further instantiated. (basic_props:not_further_inst/2)

Usage 2: `@A`

– *The following properties are added globally:*
`A` is not further instantiated. (basic_props:not_further_inst/2)

-/1: MODE

Usage 1: `-A`

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

Usage 2: `-A`

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

?/1: MODE

***/1:** MODE

+/2: MODE

Usage 1: `A+X`

– *The following properties are added at call time:*
undefined:call(X,A) (undefined property)

Usage 2: `A+X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

– *The following properties are added at call time:*
`A` is currently a term which is not a free variable. (term_typing:nonvar/1)

@/2: MODE

Usage 1: `@(A,X)`

– *The following properties are added at call time:*
undefined:call(X,A) (undefined property)

– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)

– *The following properties are added globally:*
`A` is not further instantiated. (basic_props:not_further_inst/2)

Usage 2: `@(A,X)`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

– *The following properties are added globally:*
`A` is not further instantiated. (basic_props:not_further_inst/2)

-/2: MODE

Usage 1: `A-X`

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)

Usage 2: `A-X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

?/2: MODE

Usage 1: `A?X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)

Usage 2: `A?X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

***/2:** MODE

Usage: `A*X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

in/1: MODE

Usage: `in(A)`

– *The following properties are added at call time:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

out/1: MODE

Usage: `out(A)`

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

go/1: MODE

Usage: `go(A)`

– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

in/2: MODE

Usage: `in(A,X)`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

– *The following properties are added at call time:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

– *The following properties are added upon exit:*
`A` is currently ground (it contains no variables). (term_typing:ground/1)

out/2: MODE

Usage: `out(A,X)`

- *Call and exit are compatible with:*
  undefined:call(X,A) (undefined property)
- *The following properties are added at call time:*
  `A` is a free variable. (term_typing:var/1)
- *The following properties are added upon exit:*
  `A` is currently ground (it contains no variables). (term_typing:ground/1)

go/2: MODE

Usage: `go(A,X)`

- *Call and exit are compatible with:*
  undefined:call(X,A) (undefined property)
- *The following properties are added upon exit:*
  `A` is currently ground (it contains no variables). (term_typing:ground/1)

104.4 Documentation on exports (`apll`)

initial/0: PREDICATE

Usage:

- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$start_thread/1: PREDICATE

Usage:

- *The following properties should hold at call time:*
  `Arg1` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$number_agents/1: PREDICATE

Usage:

- *The following properties should hold at call time:*
  `Arg1` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*
  `Arg1` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$push_goal/3: PREDICATE

Usage:

- *Calls should, and exit will be compatible with:*
  `Arg3` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*
  `Arg1` is an integer. (basic_props:int/1)
  `Arg2` is an integer. (basic_props:int/1)
- *The following properties hold upon exit:*
  `Arg3` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$find_goal/3: PREDICATE

Usage:

- *The following properties should hold at call time:*
  `Arg1` is an integer. (basic_props:int/1)
  `Arg2` is an integer. (basic_props:int/1)
  `Arg3` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*
  `Arg3` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_available/1: PREDICATE

Usage:

- *The following properties should hold at call time:*
  `Arg1` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$cancellation/1: PREDICATE

Usage:

- *The following properties should hold at call time:*
  `Arg1` is an integer. (basic_props:int/1)
- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$retrieve_goal/2: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

`Arg2` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`Arg2` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_det/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_det/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_nondet/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_not_executed/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_not_executed/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_rem_executing/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_rem_executing/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_finished/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_finished/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_tobacktrack/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_tobacktrack/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_toreexecute/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_toreexecute/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_failed/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_failed/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$goal_cancelled/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$show_handler/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$set_goal_cancelled/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$send_event/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$read_event/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$save_init_execution/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$save_end_execution/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$more_solutions/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$move_execution_top/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$waiting/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$suspend/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$release/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$release_remote/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$release_some_suspended_thread/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$release_all_for_unwinding/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$enter_mutex/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$enter_mutex_self/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$enter_mutex_remote/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$exit_mutex/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$exit_mutex_self/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$exit_mutex_remote/1: PREDICATE

Usage:

– *The following properties should hold at call time:*

`Arg1` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$clean_measures/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$print_measures/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$new_measure/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$not_measure/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

$incr_num_local_backtr/0: PREDICATE

Usage:

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

start_thread/1: PREDICATE

Meta-predicate with arguments: `start_thread(goal)`.

Usage: `start_thread(Goal)`

– *Description:* Executes `Goal` in a new stack set, using a new thread. Used herein to create a number of parallel agents.

– *The following properties should hold at call time:*

`Goal` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

number_agents/1: PREDICATE

Usage: `number_agents(N)`

– *Description:* Returns in `N` the number of agents in the system.

– *The following properties should hold at call time:*

`N` is a free variable. (term_typing:var/1)

`N` is an integer. (basic_props:int/1)

push_goal/3: PREDICATE

Meta-predicate with arguments: `push_goal(goal,?,?)`.

Usage: `push_goal(Goal,Det,Handler)`

– *Description:* Atomically creates the handler (an arbitrary structure in the heap) associated to `Goal`, described as deterministic or not by `Det`, and adds to the goal list a pointer to `Handler`. `Det` states whether the goal is deterministic or not.
– *The following properties should hold at call time:*

`Goal` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Det` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Goal` is an integer. (basic_props:int/1)
`Det` is an integer. (basic_props:int/1)
`Handler` is an integer. (basic_props:int/1)

push_goal/1: PREDICATE

Usage: `push_goal(Handler)`

– *Description:* Atomically adds a pointer to the particular handler `Handler` to the goal list in order to reexecute the goal associated to it.
– *The following properties should hold at call time:*

`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

find_goal/2: PREDICATE

Usage: `find_goal(GS,Handler)`

– *Description:* Searches for a goal handler `Handler` published in some goal list and succeeds if one is found, failing otherwise. The access to each goal list is made atomically.
– *The following properties should hold at call time:*

`GS` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is a free variable. (term_typing:var/1)
`GS` is an integer. (basic_props:int/1)
`Handler` is an integer. (basic_props:int/1)

find_det_goal/2: PREDICATE

Usage: `find_det_goal(GS,Handler)`

– *Description:* Searches for a deterministic goal associated to `Handler` and succeeds if one is found, failing otherwise. The access to each goal list is made atomically.
– *The following properties should hold at call time:*

`GS` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is a free variable. (term_typing:var/1)
`GS` is an integer. (basic_props:int/1)
`Handler` is an integer. (basic_props:int/1)

goal_available/1: PREDICATE

Usage: `goal_available(Handler)`

– *Description:* Succeeds if `Handler` is still in the goal list, and fails otherwise.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

cancellation/1: PREDICATE

Usage: `cancellation(Handler)`

– *Description:* Frees the memory used by `Handler`.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

retrieve_goal/2: PREDICATE

Usage: `retrieve_goal(Handler,Goal)`

– *Description:* Returns in `Goal` the parallel goal associated to `Handler`.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Goal` is a free variable. (term_typing:var/1)
`Handler` is an integer. (basic_props:int/1)
`Goal` is an integer. (basic_props:int/1)

goal_det/1: PREDICATE

Usage: `goal_det(Handler)`

– *Description:* Succeeds if the the goal associated to `Handler` is deterministic, and fails otherwise.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

set_goal_det/1: PREDICATE

Usage: `set_goal_det(Handler)`

– *Description:* Marks the goal associated to `Handler` as deterministic.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

set_goal_nondet/1: PREDICATE

Usage: `set_goal_nondet(Handler)`

– *Description:* Marks the goal associated to `Handler` as non-deterministic.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

goal_not_executed/1: PREDICATE

Usage: `goal_not_executed(Handler)`

- *Description:* Succeeds if the goal associated to `Handler` has not been executed yet, and fails otherwise.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

set_goal_not_executed/1: PREDICATE

Usage: `set_goal_not_executed(Handler)`

- *Description:* Sets the execution of the goal associated to `Handler` to never executed.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

goal_rem_executing/1: PREDICATE

Usage: `goal_rem_executing(Handler)`

- *Description:* Succeeds if the goal associated to `Handler` is remotely executing, and fails otherwise.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

set_goal_rem_executing/1: PREDICATE

Usage: `set_goal_rem_executing(Handler)`

- *Description:* Sets the execution of the goal associated to `Handler` to being remotely executing.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

goal_finished/1: PREDICATE

Usage: `goal_finished(Handler)`

- *Description:* Succeeds if the execution of the goal associated to `Handler` has finished, and fails otherwise.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

set_goal_finished/1: PREDICATE

Usage: `set_goal_finished(Handler)`

- *Description:* Sets the execution of the goal associated to `Handler` to finished.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

goal_tobacktrack/1: PREDICATE

Usage: `goal_tobacktrack(Handler)`

- *Description:* Succeeds if the execution of the goal associated to `Handler` has to backtrack, and fails otherwise.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

set_goal_tobacktrack/1: PREDICATE

Usage: `set_goal_tobacktrack(Handler)`

- *Description:* Sets the execution of the goal associated to `Handler` to backtrack.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

goal_toreexecute/1: PREDICATE

Usage: `goal_toreexecute(Handler)`

- *Description:* Succeeds if the execution of the goal associated to `Handler` has to backtrack, and fails otherwise.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

set_goal_toreexecute/1: PREDICATE

Usage: `set_goal_toreexecute(Handler)`

- *Description:* Sets the execution of the goal associated to `Handler` to be reexecuted.
- *The following properties should hold at call time:*

  `Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Handler` is an integer. (basic_props:int/1)

goal_failed/1: PREDICATE

Usage: `goal_failed(Handler)`

- *Description:* Succeeds if the execution of the goal associated to `Handler` has failed, and fails otherwise.

– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

set_goal_failed/1: PREDICATE

Usage: `set_goal_failed(Handler)`

– *Description:* Sets the execution of the goal associated to `Handler` to failed.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

show_handler/1: PREDICATE

Usage: `show_handler(Handler)`

– *Description:* Succeeds if the execution of the goal associated to `Handler` has cancelled, and fails otherwise.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

goal_cancelled/1: PREDICATE

Usage: `goal_cancelled(Handler)`

– *Description:* Succeeds if the execution of the goal associated to `Handler` has cancelled, and fails otherwise.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

set_goal_cancelled/1: PREDICATE

Usage: `set_goal_cancelled(Handler)`

– *Description:* Sets the execution of the goal associated to `Handler` to cancelled.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

send_event/1: PREDICATE

Usage: `send_event(Handler)`

– *Description:* Sends the handler to the agent that picked up the goal associated to the handler `Handler` in order to perform backtracking over it.
– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

read_event/1: PREDICATE

Usage: `read_event(Handler)`

– *Description:* Succeeds if the event queue of the agent is not empty and unifies `Handler` with the handler associated to the goal to backtrack over.

– *The following properties should hold at call time:*

`Handler` is a free variable. (term_typing:var/1)

`Handler` is an integer. (basic_props:int/1)

save_init_execution/1: PREDICATE

Usage: `save_init_execution(Handler)`

– *Description:* Saves the choice point that marks the starting point of the execution of the parallel goal associated to the handler `Handler`.

– *The following properties should hold at call time:*

`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Handler` is an integer. (basic_props:int/1)

save_end_execution/1: PREDICATE

Usage: `save_end_execution(Handler)`

– *Description:* Saves the choice point that marks the final point of the execution of the parallel goal associated to the handler `Handler`.

– *The following properties should hold at call time:*

`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Handler` is an integer. (basic_props:int/1)

more_solutions/1: PREDICATE

Usage: `more_solutions(Handler)`

– *Description:* Succeeds whether the goal associated to `Handler` has more solutions to compute, and fails otherwise.

– *The following properties should hold at call time:*

`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Handler` is an integer. (basic_props:int/1)

move_execution_top/1: PREDICATE

Usage: `move_execution_top(Handler)`

– *Description:* Moves the choice point of the goal saved into the handler `Handler` to the top of the stack.

– *The following properties should hold at call time:*

`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Handler` is an integer. (basic_props:int/1)

waiting/1: PREDICATE

Usage: `waiting(Handler)`

– *Description:* Succeeds when the execution of the publishing agent associated to `Handler` is suspended, and fails otherwise.

– *The following properties should hold at call time:*

`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Handler` is an integer. (basic_props:int/1)

suspend/0: PREDICATE

Usage:

– *Description:* Suspends the execution of the current thread.

release/1: PREDICATE

Usage: `release(Handler)`

– *Description:* Releases the execution of the publishing agent associated to `Handler`.

– *The following properties should hold at call time:*

`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Handler` is an integer. (basic_props:int/1)

release_remote/1: PREDICATE

Usage: `release_remote(Handler)`

– *Description:* Releases the execution of the agent that picked up the goal associated to `Handler`.

– *The following properties should hold at call time:*

`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Handler` is an integer. (basic_props:int/1)

release_some_suspended_thread/0: PREDICATE

Usage:

– *Description:* Selects one out of any suspended threads and sends it a signal to resume its execution.

release_all_for_unwinding/0: PREDICATE

Usage:

– *Description:* Releases the execution of all agents to perform stack unwinding.

enter_mutex/1: PREDICATE

Usage: `enter_mutex(Handler)`

– *Description:* Attemps to enter into a mutual exclusion to access shared variables of the publishing agent associated to `Handler`.

– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

enter_mutex_self/0: PREDICATE

Usage:

– *Description:* The mutual exclusion is to access the shared variables of the calling thread.

enter_mutex_remote/1: PREDICATE

Usage: `enter_mutex_remote(Handler)`

– *Description:* The mutual exclusion is to access the shared variables of the remote thread associated to `Handler`.

– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

exit_mutex/1: PREDICATE

Usage: `exit_mutex(Handler)`

– *Description:* Exits from the mutual exclusion of the publishing agent associated to `Handler`.

– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

exit_mutex_self/0: PREDICATE

Usage:

– *Description:* Exits from the local mutual exclusion.

exit_mutex_remote/1: PREDICATE

Usage: `exit_mutex_remote(Handler)`

– *Description:* Exits from the mutual exclusion of the remote agent associated to `Handler`.

– *The following properties should hold at call time:*
`Handler` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Handler` is an integer. (basic_props:int/1)

clean_measures/0: PREDICATE

Usage:

– *Description:* Restarts the statistical measures for nondeterministic parallel programs.

print_measures/0: PREDICATE

Usage:

– *Description:* Prints the value of the statistical measures for nondeterministic parallel goals.

new_measure/0: PREDICATE

Usage:

– *Description:* Prepares the statistical values for a new execution.

not_measure/0: PREDICATE

Usage:

– *Description:* Avoids measuring.

incr_num_local_backtr/0: PREDICATE

Usage:

– *Description:* Increments the number of times that backwards execution has been performed over nondeterministic parallel goals.

105 Call on determinate

Author(s): Jose F. Morales, Manuel Carro.

Offers an enriched variant of call and cut `!!/0` which executes pending goals when the computation has no more alternatives.

This library is useful to, for example, get rid of external connections once the necessary data has been obtained.

105.1 Usage and interface (`det_hook_doc`)

- **Library usage:**
  `:- use_module(library(det_hook_rt)).`
  in which case, `!!/0` is not available.
  Typically, this library is used as a package:
  `:- use_package(det_hook).`
- **New operators defined:**
  `?/1` [200,fy], `@/1` [200,fy].
- **New modes defined:**
  `+/1`, `@/1`, `-/1`, `?/1`, `*/1`, `+/2`, `@/2`, `-/2`, `?/2`, `*/2`.
- **Other modules used:**
  - *System library modules:*
    `det_hook/det_hook_rt`.

105.2 Documentation on new modes (`det_hook_doc`)

`+/1`: MODE

Usage: `+A`

- *The following properties are added at call time:*
  `A` is currently a term which is not a free variable. (term_typing:nonvar/1)

`@/1`: MODE

Usage: `@A`

- *The following properties are added globally:*
  `A` is not further instantiated. (basic_props:not_further_inst/2)

`-/1`: MODE

Usage: `-A`

- *The following properties are added at call time:*
  `A` is a free variable. (term_typing:var/1)

?/1: MODE

***/1:** MODE

+/2: MODE

Usage: `A+X`

– *The following properties are added at call time:*
undefined:call(X,A) (undefined property)

@/2: MODE

Usage: `@(A,X)`

– *The following properties are added at call time:*
undefined:call(X,A) (undefined property)

– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)

– *The following properties are added globally:*
`A` is not further instantiated. (basic_props:not_further_inst/2)

-/2: MODE

Usage: `A-X`

– *The following properties are added at call time:*
`A` is a free variable. (term_typing:var/1)

– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)

?/2: MODE

Usage: `A?X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

– *The following properties are added upon exit:*
undefined:call(X,A) (undefined property)

***/2:** MODE

Usage: `A*X`

– *Call and exit are compatible with:*
undefined:call(X,A) (undefined property)

105.3 Other information (det_hook_doc)

As an example, the program

```
:- module(_, _, [det_hook]).

enumerate(X):-
        display(enumerating), nl,
        OnCut = (display('goal cut'), nl),
        OnFail = (display('goal failed'), nl),
        det_try(enum(X), OnCut, OnFail).

enum(1).
enum(2).
enum(3).
```

behaves as follows:

```
?- enumerate(X).
enumerating

X = 1 ? ;

X = 2 ? ;

X = 3 ? ;
goal failed
```

(note the message inserted on failure). The execution can be cut as follows:

```
?- use_package(det_hook).
{Including /home/clip/lib/ciao/ciao-1.7/library/det_hook/det_hook.pl
}

yes
?- enumerate(X), '!!'.
enumerating
goal cut

X = 1 ? ;

no
```

105.4 Known bugs and planned improvements (det_hook_doc)

- If the started goals do not exhaust their solutions, and '!!'/0 is not used, the database will populate with facts which will be consulted the next time a '!!'/0 is used. This could cause incorrect executions.

106 Runtime predicates for call on determinate

Author(s): Jose F. Morales, Manuel Carro.

Implementation of variant of call and cut which executes pending goals when the computation has no more alternatives.

106.1 Usage and interface (`det_hook_rt`)

- **Library usage:**
  `:- use_module(library(det_hook_rt)).`
- **Exports:**
  - *Predicates:*
    `det_try/3`.

106.2 Documentation on exports (`det_hook_rt`)

det_try/3: PREDICATE

Meta-predicate with arguments: `det_try(goal,goal,goal)`.

Usage: `det_try(Goal,OnCut,OnFail)`

- *Description:* `Action` is called, and `OnCut` and `OnFail` are goals to be executed when `Goal` is cut or when it finitely fails, respectively. In order for this to work, cutting must be performed in a special way, by using the `!!/0` predicate, also provided by this module.
- *The following properties should hold at call time:*
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  `OnCut` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  `OnFail` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

107 Miscellaneous predicates

Author(s): Manuel Carro, Daniel Cabeza.

This module implements some miscellaneous non-logical (but sometimes very useful) predicates.

107.1 Usage and interface (odd)

- **Library usage:**
  `:- use_module(library(odd)).`
- **Exports:**
  - *Predicates:*
    `setarg/3`, `undo/1`.

107.2 Documentation on exports (odd)

setarg/3: PREDICATE

Usage: `setarg(Index,Term,NewArg)`

- *Description:* Replace destructively argument `Index` in `Term` by `NewArg`. The assignment is undone on backtracking. This is a major change to the normal behavior of data assignment in Ciao Prolog.
- *The following properties should hold at call time:*

  `Index` is currently instantiated to an integer. (term_typing:integer/1)

  `Term` is a compound term. (basic_props:struct/1)

  `NewArg` is any term. (basic_props:term/1)

undo/1: PREDICATE

Meta-predicate with arguments: `undo(goal)`.

Usage: `undo(Goal)`

- *Description:* `call(Goal)` is executed on backtracking. This is a major change to the normal control of Ciao Prolog execution.
- *The following properties should hold at call time:*

  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
- *The following properties should hold upon exit:*

  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

108 Delaying predicates (freeze)

Author(s): Remy Haemmerle, Manuel Carro, Daniel Cabeza.

This library offers a simple implementation of `freeze/2`, `frozen/2`, etc. [Col82,Nai85,Nai91,Car87] based on the use of attributed variables [Hol92,Hol90].

108.1 Usage and interface (`freeze`)

- **Library usage:**
  `:- use_module(library(freeze)).`
- **Exports:**
  - *Predicates:*
    `freeze/2`, `frozen/2`.

108.2 Documentation on exports (`freeze`)

freeze/2: PREDICATE

Meta-predicate with arguments: `freeze(?,goal)`.

Usage: `freeze(X,Goal)`

- *Description:* If `X` is free delay `Goal` until `X` is non-variable.
- *The following properties should hold at call time:*
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

frozen/2: PREDICATE

Meta-predicate with arguments: `frozen(?,goal)`.

Usage: `frozen(X,Goal)`

- *Description:* `Goal` is currently delayed until variable `X` becomes bound.
- *The following properties should hold upon exit:*
  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

108.3 Known bugs and planned improvements (`freeze`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

109 Delaying predicates (when)

Author(s): Manuel Carro.

`when/2` delays a predicate until some condition in its variable is met. For example, we may want to find out the maximum of two numbers, but we are not sure when they will be instantiated. We can write the standard `max/3` predicate (but changing its name to `gmax/3` to denote that the first and second arguments must be ground) as

```
gmax(X, Y, X):- X > Y, !.
gmax(X, Y, Y):- X =< Y.
```

and then define a 'safe' `max/3` as

```
max(X, Y, Z):-
        when((ground(X),ground(Y)), gmax(X, Y, Z)).
```

which can be called as follows:

```
?- max(X, Y, Z) , Y = 0, X = 8.

X = 8,
Y = 0,
Z = 8 ?

yes
```

Alternatively, `max/3` could have been defined as

```
max(X, Y, Z):-
        when(ground((X, Y)), gmax(X, Y, Z)).
```

with the same effects as above. More complex implementations are possible. Look, for example, at the `max.pl` implementation under the `when` library directory, where a `max/3` predicate is implemented which waits on all the arguments until there is enough information to determine their values:

```
?- use_module(library(when(max))).

yes
?- max(X, Y, Z), Z = 5, Y = 4.

X = 5,
Y = 4,
Z = 5 ?

yes
```

109.1 Usage and interface (when)

- **Library usage:**
  `:- use_module(library(when)).`
- **Exports:**
  - *Predicates:*
    when/2.
  - *Regular Types:*
    wakeup_exp/1.
- **Other modules used:**
  - *System library modules:*
    terms_vars, sort, sets.

109.2 Documentation on exports (when)

when/2: PREDICATE

Meta-predicate with arguments: `when(?,goal)`.

Usage: `when(WakeupCond,Goal)`

- *Description:* Delays / executes `Goal` according to `WakeupCond` given. The `WakeupCond`s now acceptable are `ground(T)` (`Goal` is delayed until `T` is ground), `nonvar(T)` (`Goal` is delayed until `T` is not a variable), and conjunctions and disjunctions of conditions:

```
wakeup_exp(ground(_1)).
wakeup_exp(nonvar(_1)).
wakeup_exp((C1,C2)) :-
        wakeup_exp(C1),
        wakeup_exp(C2).
wakeup_exp((C1;C2)) :-
        wakeup_exp(C1),
        wakeup_exp(C2).
```

  `when/2` only fails it the `WakeupCond` is not legally formed. If `WakeupCond` is met at the time of the call no delay mechanism is involved — but there exists a time penalty in the condition checking.

  In case that an instantiation fires the execution of several predicates, the order in which these are executed is not defined.

- *The following properties should hold at call time:*

  `WakeupCond` is a legal expression for delaying goals. (when:wakeup_exp/1)

  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

wakeup_exp/1: REGTYPE

Usage: `wakeup_exp(T)`

- *Description:* `T` is a legal expression for delaying goals.

109.3 Known bugs and planned improvements (`when`)

- Redundant conditions are not removed.
- Floundered goals are not appropriately printed.

110 Active modules (high-level distributed execution)

Author(s): Manuel Hermenegildo, Daniel Cabeza.

Active modules [CH95] provide a high-level model of inter-process communication and distributed execution (note that this is also possible using Ciao's communication and concurrency primitives, such as sockets, concurrent predicates, etc., but at a lower level of abstraction). An *active module* (or an *active object*) is an ordinary module to which computational resources are attached, and which resides at a given location on the network. Compiling an active module produces an executable which, when running, acts as a *server* for a number of predicates: the predicates exported by the module. Predicates exported by an active module can be accessed by a program on the network by simply "using" the module, which then imports such "remote predicates." The process of "using" an active module does not involve transferring any code, but rather setting up things so that calls in the module using the active module are executed as remote procedure calls to the active module. This occurs in the same way independently of whether the active module and the using module are in the same machine or in different machines across the network.

Except for having to compile it in a special way (see below), an active module is identical from the programmer point of view to an ordinary module. A program using an active module imports it and uses it in the same way as any other module, except that it uses "`use_active_module`" rather than "`use_module`" (see below). Also, an active module has an address (network address) which must be known in order to use it. In order to use an active module it is necessary to know its address: different "protocols" are provided for this purpose (see below).[1]

From the implementation point of view, active modules are essentially daemons: executables which are started as independent processes at the operating system level. Communication with active modules is implemented using sockets (thus, the address of an active module is an IP socket address in a particular machine). Requests to execute goals in the module are sent through the socket by remote programs. When such a request arrives, the process running the active module takes it and executes it, returning through the socket the computed answers. These results are then taken and used by the remote processes. Backtracking over such remote calls works as usual and transparently. The only limitation (this may change in the future, but it is currently done for efficiency reasons) is that all alternative answers are precomputed (and cached) upon the first call to an active module and thus *an active module should not export a predicate which has an infinite number of answers.*

The first thing to do is to select a method whereby the client(s) (the module(s) that will use the active module) can find out in which machine/port (IP address/socket number) the server (i.e., the active module) will be listening once started, i.e., a "protocol" to communicate with the active module. The easiest way to do this is to make use of the rendezvous methods which are provided in the Ciao distribution in the `library/actmods` directory; currently, `tmpbased...`, `filebased...`, `webbased...`, and `platformbased...`.

The first one is based on saving the IP address and socket number of the server in a file in a predefined directory (generally `/tmp`, but this can be changed by changing `tmpbased_common.pl`).

The second one is similar but saves the info in the directory in which the server is started (as `<`*module_name*`>.addr`), or in the directory that a `.addr` file, if it exists, specifies. The clients must be started in the same directory (or have access to a file `.addr` specifying the same directory). However, they can be started in different machines, provided this directory is shared (e.g., by NFS or Samba), or the file can be moved to an appropriate directory on a different machine –provided the full path is the same.

[1] It is also possible to provide active modules via a WWW address. However, we find it more straightforward to simply use socket addresses. In any case, this is generally hidden inside the access method and can be thus made transparent to the user.

The third one is based on a name server for active modules. When an active module is started, it communicates its address to the name server. When the client of the active module wants to communicate with it, it asks the name server the active module address. This is all done transparently to the user. The name server must be running when the active module is started (and, of course, when the application using it is executed). The location of the name server for an application must be specified in an application file named `webbased_common.pl` (see Section 3.1 below).

The fourth one is also based on a name server, but the address of the name server is given as a parameter to the active modules when started.

The rendezvous methods (or protocols) are encoded in two modules: a first one, called `...publish.pl`, is used by the server to publish its info. The second one, called `...locate.pl`, is used by the client(s) to locate the server info. For efficiency, the client methods maintain a cache of addresses, so that the server information only needs to be read from the file system the first time the active module is accessed.

Active modules are compiled using the `-a` option of the Ciao compiler (this can also be done from the interactive top-level shell using `make_actmod/2`). For example, issuing the following command:

```
ciaoc -a 'actmods/filebased_publish' simple_server
```

compiles the simple server example that comes with the distribution (in the `actmods/example` directory). The `simple_client_with_main` example (in the same directory) can be compiled as usual:

```
ciaoc simple_client_with_main
```

Note that the client uses the `actmods` package, specifies the rendezvous method by importing `library(actmods(filebased_locate))`, and explicitely imports the "remote" predicates (*implicit imports will not work*). Each module using the `actmods` package *should only use one of the rendezvous methods.*

Now, if the server is running (e.g., `simple_server &` in Un*x or double-clicking on it in Win32) when the client is executed it will connect with the server to access the predicate(s) that it imports from it.

A simpler even client `simple_client.pl` can be loaded into the top level and its predicates called as usual (and they will connect with the server if it is running).

110.1 Active modules as agents

It is rather easy to turn Ciao active modules into agents for some kind of applications. The directory `examples/agents` contains a (hopefully) self-explanatory example.

110.2 Usage and interface (`actmods_doc`)

- **Library usage:**
  `:- use_package(actmods).`
  or
  `:- module(...,...,[actmods]).`
- **New declarations defined:**
  `use_active_module/2`.

110.3 Documentation on new declarations (actmods_doc)

use_active_module/2: DECLARATION

Usage: :- `use_active_module(AModule,Imports).`

- *Description:* Specifies that this code imports from the *active module* defined in `AModule` the predicates in `Imports`. The imported predicates must be exported by the active module.
- *The following properties should hold at call time:*

  `AModule` is a source name. (streams_basic:sourcename/1)

  `Imports` is a list of `prednames`. (basic_props:list/2)

110.4 Other information (actmods_doc)

The protocols `webbased` and `platformbased` are described in this section with a bit more detail.

110.5 Active module name servers (webbased protocol)

An application using a name server for active modules must have a file named `webbased_common.pl` that specifies where the name server resides. It must have the `URL` and the path which corresponds to that `URL` in the file system of the server machine (the one that hosts the `URL`) of the file that will hold the name server address.

The current distribution provides a file `webbased_common.pl` that can be used (after proper setting of its contents) for a server of active modules for a whole installation. Alternatively, particular servers for each application can be set up (see below).

The current distribution also provides a module that can be used as name server by any application. It is in file `examples/webbased_server/webbased_server.pl`.

To set up a name server edit `webbased_common.pl` to change its contents appropriately as described above (`URL` and corresponding complete file path). Then recompile this library module:

```
ciaoc -c webbased_common
```

The name server has to be compiled as an active module itself:

```
ciaoc -a actmods/webserver_publish webbased_server
```

It has to be started in the server machine before the application and its active modules are compiled.

Alternatively, you can copy `webbased_common.pl` and use it to set up name servers for particular applications. Currently, this is a bit complicated. You have to ensure that the name server, the application program, and all its active modules are compiled and executed with the same `webbased_common.pl` module. One way to do this is to create a subdirectory `actmods` under the directory of your application, copy `webbased_common.pl` to it, modify it, and then compile the name server, the application program, and its active modules using a library path that guarantees that your `actmods` directory is located by the compiler before the standard Ciao library. The same applies for when running all of them if the library loading is dynamic.

One way to do the above is using the `-u` compiler option. Assume the following file:

```
:- module(paths,[],[]).
:- multifile library_directory/1.
:- dynamic library_directory/1.
```

```
:- initialization(asserta_fact(
library_directory('/root/path/to/my/particular/application') )).
```

then you have file `webbased_common.pl` in a subdirectory `actmods` of the above cited path. You have to compile the name server, the active modules, and the rest of the application with:

```
ciaoc -u paths -s ...
```

to use your particular `webbased_common.pl` and to make executables statically link libraries. If they are dynamic, then you have to provide for the above library_directory path to be set up upon execution. This can be done, for example, by including module `paths` into your executables.

Addresses of active modules are saved by the name server in a subdirectory `webbased_db` of the directory where you start it —see `examples/webbased_server/webbased_db/webbased_server`). This allows to restart the server right away if it dies (since it saves its state). This directory should be cleaned up regularly of addresses of active modules which are no more active. To do this, stop the server —by killing it (its pid is in `PATH/FILE`), and restart it after cleaning up the files in the above mentioned directory.

110.6 Platforms (platformbased protocol)

This protocol is also based on a name server. There are, however, two differences with the above one: the name server address and the active modules names are dynamic. On the one hand, the name server address (IP address/socket number) is given to the active modules when they are started up. This might be convenient when using the same name server executable for different applications starting up a different name server process for each application. On the other hand, the name assigned to a given active module can also be given as a parameter to the active module when it is started up. This makes it easier to maintain a local name space for particular applications (e.g., two modules with the same name can be used as active modules in the same application).

The code of a name server for the previous section protocol can also be used for this protocol (e.g., file `examples/webbased_server/webbased_server.pl`).

110.7 Known bugs and planned improvements (`actmods_doc`)

- The package provides no means for security: the accessing application must take care of this (?).
- It can happen that there is a unique process for an active module serving calls from several different simultaneous executions of the same application (or even different applications). In this case, there might be unwanted interactions (e.g., if the active module has state).
- Applications may fail if the name server or an active module is restarted during execution of the application (since they might restart at a different port than the one cached by the application).
- One may want name servers to reside at a fixed and known machine and port number (this is known as a *service* and is defined in `/etc/services` in a Un*x machine).

111 Agents

Author(s): Francisco Bueno.

An agent is an active module which has a main execution thread. Simultaneously (i.e., in concurrent execution with the main thread), the agent receives messages from other agents, which trigger the execution of a predicate by the name of the message. Messages can also be sent to other agents, by calling the predicate by the name of the message in the context of the receiver agent (see `::/2` below). Agents are identified by name. The name of an agent is usually the name of its (main) file, but this depends on the protocol used (see `protocol/1` below).

A simple agent that sends inform messages and at the same time receives them from other agents, answering back ok, will look like:

```
:- agent(simple,[inform/2,ok/1]).
:- protocol('actmods/filebased').

agent :-
    repeat,
    display('Agent id:message?- '), read(Agent:Mess),
    Agent::inform(Mess),
    fail.

inform(Agent,Mess):-
    display(Agent), display(' has sent: ', display(Mess), nl,
    Agent::ok.

ok(_Agent).
```

111.1 Usage and interface (agent_doc)

- **Library usage:**
  `:- agent(AgentName,[Message|...]).`
  for the main file of the agent.
  `:- use_module(library(agent(agent_call))).`
  for the rest of modules of the agent that need to send messages.
- **New operators defined:**
  `::/2` [550,xfx].
- **New declarations defined:**
  `protocol/1`.

111.2 Documentation on new declarations (agent_doc)

protocol/1: DECLARATION

A protocol is formed by a pair of modules which allow to locate connection addresses of agents. By convention, the names of these modules have a common prefix, which makes reference to the protocol, and have suffixes '_locate' and '_publish'. The 'publish' part of the protocol must define a multifile predicate `save_addr_actmod/1` and the 'locate' part export a predicate `module_address/2`. The first one publishes an agent address; the

second one locates the address of an agent. Together, both make it possible for agents to send and receive messages. All agents in a multi-agent system must therefore use the same protocol. Upon compilation, they will then be (automatically) instrumented as active modules under the corresponding rendezvous method.

Usage: `:- protocol(Protocol).`

– *Description:* `Protocol` is the prefix to a library path where an active module rendezvous protocol can be found.

111.3 Documentation on multifiles (agent_doc)

save_addr_actmod/1: PREDICATE

The predicate is *multifile*.

Usage: `save_addr_actmod(Address)`

– *Description:* (protocol defined) publishes the agent's `Address`.

111.4 Documentation on internals (agent_doc)

module_address/2: PREDICATE

Usage: `module_address(Agent,Address)`

– *Description:* (protocol defined) gives the `Address` of `Agent`.

::/2: PREDICATE

No further documentation available for this predicate.

self/1: PREDICATE

No further documentation available for this predicate.

111.5 Other information (agent_doc)

This package is intended as a sample of how to program agents in Ciao, based on active modules. It probably lacks many features that an agent might need. In particular, it lacks language-independence: it is thought for multi-agent systems where all agents are programmed in Ciao.

You are welcome to add any feature that you may be missing!

111.5.1 Platforms

A platform is an active module which holds connection addresses of agents in a multi-agent system. A protocol is provided which enables the use of platforms: `agent/platformbased`. A suitable platform must be up and running when agents which run under this protocol are started up. The host id and port number (IP address/socket number) of the platform must then be given as arguments to the agents executables. The protocol also allows to give an agent name to the agent upon start-up. A module suitable for a platform can be found in `library(actmods/examples/webbased_server/webbased_server)`.

111.6 Known bugs and planned improvements (`agent_doc`)

- Currently, the agent has to be compiled explicitely as an active module. The same protocol than in the agent source code must be used for this. Automatic compilation is not working.
- It seems that there are running-ahead problems with threads that prevent to correctly publish agent addresses, sometimes.

112 Breadth-first execution

Author(s): Daniel Cabeza, Manuel Carro, Manuel Hermenegildo.

This package implements breadth-first execution of predicates. This may be useful in search problems when a proof procedure is needed that will find all solutions (even if it may still loop for some failures). This is in contrast with the default depth-first search, which may loop in some cases even if there are correct answers to a given query. This library is also useful when experimenting with pure programs as well as when teaching logic programming, for illustrating the expected theoretical results that should be expected from the declarative semantics (see for example the slides in `http://www.cliplab.org/proglog`).

It is important to realize, however, that the improved behaviour of breadth first execution comes at a high (exponential!) price in terms of both time ad memory. This library allows the programmer to control this overhead by selecting which predicates will be executed in breadth-first mode and which predicates in depth-first mode. More concretely, predicates written with operators `'<-'/1` (facts) and `'<-'/2` (clauses) are executed using breadth-first search, while predicates using the standard syntax will be executed depth-first.

The following example implements two versions of a predicate meant to succeed if two nodes of a directed graph are connected. The `chain/2` predicate (which will be executed depth-first) loops without finding the connection between `a` and `d`, while the `bfchain/2` predicate (which will be executed breadth-first) will find the connection correctly:

```
:- module(chain, _, [bf]).

test(bf) :- bfchain(a,d).
test(df) :- chain(a,d).   % loops!

bfchain(X,X) <- .
bfchain(X,Y) <- arc(X,Z), bfchain(Z,Y).

chain(X,X).
chain(X,Y) :- arc(X,Z), chain(Z,Y).

arc(a,b).
arc(a,d).
arc(b,c).
arc(c,a).
```

A second package, `'bf/bfall'`, allows executing *all* the predicates in a given module in breadth-first mode. In this case, predicates should be written using the standard syntax. This is useful to be able to switch easily between depth-first and breadth-first execution (e.g., for testing purposes) for all predicates in a given module without having to modify the program. The following program (written in standard syntax) runs breadth-first:

```
:- module(chain_bfall, _, ['bf/bfall']).

test :- chain(a,d).

chain(X,X).
chain(X,Y) :- arc(X,Z), chain(Z,Y).

arc(a,b).
arc(a,d).
arc(b,c).
```

```
arc(c,a).
```

There is another version, package `'bf/af'`, which ensures AND-fairness by goal shuffling. This reduces the number of cases in which an execution that is a failure loops instead (infinite failures) at a small additional cost. For example, by using `'bf/af'` the following code correctly answers "`no`" when executing `test/0`:

```
:- module(sublistapp, [test/0, sublistapp/2], ['bf/af']).

:- push_prolog_flag(unused_pred_warnings, no).

test :- sublistapp([a], [b]).

sublistapp(S, L) <- append(_, S, Y), append(Y, _, L).

append([], L, L) <- .
append([X|Xs], L, [X|Ys]) <- append(Xs, L, Ys).

:- pop_prolog_flag(unused_pred_warnings).
```

There is also a package `'bf/bfall'` which again allows executing *all* the predicates in a given module in breadth-first, and-fair mode, where also all predicates should be written using the standard syntax. This package offers (at a cost, of course) very nice results for many programs, and is used extensively in programming courses by the Ciao developers.

Finally, it should be noted that a separate library, `id`, implements *iterative-deepening* search, which can in many cases be a better alternative to breadth-first search, since it achieves the same improvement in the completeness results in many cases at a greatly reduced execution cost (but the enumeration order of solutions is not as nice, and that is why these packages are very attractive for prototyping and teaching).

112.1 Usage and interface (`bf_doc`)

- **Library usage:**
  `:- use_package(bf).`
  or
  `:- module(...,...,[bf]).`
- **New operators defined:**
  `<-/2` [1200,xfx], `<-/1` [1200,xf].

112.2 Known bugs and planned improvements (`bf_doc`)

- Does not correctly work in user files.

113 Iterative-deepening execution

Author(s): Rémy Haemmerlé, Manuel Carro, Claudio Vaucheret, Manuel Hermenegildo.

This package applies a *compiling control* technique to implement *depth first iterative deepening* execution [Kor85]. It changes the usual *depth-first* computation rule by *iterative-deepening* on those predicates specifically marked. This is very useful in search problems when a complete proof procedure is needed.

When this computation rule is used, first all goals are expanded only up to a given depth. If no solution is found or more solutions are needed by backtracking, the depth limit is incremented and the whole goal is repeated. Although it might seem that this approach is very inefficient because all higher levels are repeated for the deeper ones, it has been shown that is performs only about b/(b - 1) times as many operations than the corresponding breadth-first search, (where b is the branching factor of the proof tree) while the waste of memory is the same as depth first.

The usage is by means of the following directive:

```
:- iterative(Name, FirstCut, Formula).
```

which states than the predicate 'Name' given in functor/arity form will be executed using iterative deepening rule starting at the depth 'FirstCut' with depth being incremented by the predicate 'Formula'. This predicate compute the new depth using the previous one. It must implement a dilating function i.e. the new depth must be greater. For example, to start with depth 5 and increment by 10 you can write:

```
:- iterative(p/1,5,f).
f(X,Y) :- Y is X + 10.
```

or if you prefer,

```
:- iterative(p/1,5,(_(X,Y):- Y is X + 10)).
```

You can also use a fourth parameter to set a limiting depth. All goals below the given depth limit simply fail. Thus, with the following directive:

```
:- iterative(p/1,5,(_(X,Y):- Y is X + 10),100).
```

all goals deeper than 100 will fail.

An example of code using this package would be:

```
:- module(example_id, _,[id]).

test(id) :-
        idchain(a,d).
test(df) :-
        chain(a,d).   % loops!

:- iterative(idchain/2, 3, ( _(X,Z) :- Z is X + 1) ).

idchain(X,X).
idchain(X,Y) :-
        arc(X,Z),
        idchain(Z,Y).

chain(X,X).
chain(X,Y) :-
        arc(X,Z),
        chain(Z,Y).

arc(a,b).
```

```
arc(a,d).
arc(b,c).
arc(c,a).
```

The order of solutions are first the shallower and then the deeper. Solutions which are between two cutoff are given in the usual left to right order. For example,

```
:- module(_,_,[id]).

% All goals deeper than 2 will fail
:- iterative(p/1,0,(_(X,Z) :- Z is X + 1),2).

% Change the solutions' order to goal p(X).
%:- iterative(p/1,1,(_(X,Z) :- Z is X + 3)).

p(X) :- q(X).
p(a).

q(X) :- r(X).
q(b).

r(X) :- s(X).
r(c).

s(d).
```

Another complete proof procedure implemented is the `bf` package (breadth first execution).

113.1 Usage and interface (`id_doc`)

- **Library usage:**

  ```
  :- use_package(id).
  ```

  or

  ```
  :- module(...,...,[id]).
  ```

114 Constraint programming over rationals

Author(s): Christian Holzbaur, Daniel Cabeza, Samir Genaim (Meta-programming predicates).

Note: This package is currently being adapted to the new characteristics of the Ciao module system. This new version works right now with limitations, but it is under further development at the moment. Use with (lots of) caution.

114.1 Usage and interface (`clpq_doc`)

- **Library usage:**

  `:- use_package(clpq).`

  or

  `:- module(...,...,[clpq]).`
- **New operators defined:**

  `.=./2` [700,xfx], `.<>./2` [700,xfx], `.<./2` [700,xfx], `.=<./2` [700,xfx], `.>./2` [700,xfx], `.>=./2` [700,xfx].

114.2 Other information (`clpq_doc`)

114.2.1 Some CLP(Q) examples

(Other examples can be found in the source and library directories.)

- 'Reversible' Fibonacci (clpq):

```
:- module(_, [fib/2], []).
:- use_package(clpq).

fib(X,Y):- X .=. 0, Y .=. 0.
fib(X,Y):- X .=. 1, Y .=. 1.
fib(N,F) :-
        N .>. 1,
        N1 .=. N - 1,
        N2 .=. N - 2,
        fib(N1, F1),
        fib(N2, F2),
        F .=. F1+F2.
```

- Matrix multiplication (clpq):

```
:- use_package(clpq).
:- use_module(library(write)).

mmultiply([],_,[]).
mmultiply([V0|Rest], V1, [Result|Others]):-
            mmultiply(Rest, V1, Others),
                multiply(V1,V0,Result).
```

```
multiply([],_,[]).
multiply([V0|Rest], V1, [Result|Others]):-
            multiply(Rest, V1, Others),
                vmul(V0,V1,Result).

vmul([],[],0).
vmul([H1|T1], [H2|T2], Result):-
        vmul(T1,T2, Newresult),
        Result .=. H1*H2+Newresult.

matrix(1,[[1,2,3,4,5],[4,0,-1,5,6],[7,1,-2,8,9],[-1,0,1,3,2],[1,5,-3,2,4]]).
matrix(2,[[3,2,1,0,-1],[-2,1,3,0,2],[1,2,0,-1,5],[1,3,2,4,5],[-5,1,4,2,2]]).

%% Call with: ?- go(M).

go(M):-
        matrix(1,M1),
        matrix(2,M2),
        mmultiply(M1, M, M2).
```

- Queens (clpq):

```
:- use_package(clpq).

queens(N, Qs) :- constrain_values(N, N, Qs), place_queens(N, Qs).

constrain_values(0, _N, []).
constrain_values(N, Range, [X|Xs]) :-
        N .>. 0, X .>. 0, X .=<. Range,
        N1 .=. N - 1,
        constrain_values(N1, Range, Xs), no_attack(Xs, X, 1).

no_attack([], _Queen, _Nb).
no_attack([Y|Ys], Queen, Nb) :-
        Queen .<>. Y+Nb,
        Queen .<>. Y-Nb,
        Nb1 .=. Nb + 1,
        no_attack(Ys, Queen, Nb1).

place_queens(0, _).
place_queens(N, Q) :-
        N > 0, member(N, Q), N1 is N-1, place_queens(N1, Q).
```

114.2.2 Meta-programming with CLP(Q)

The implementation of CLP(Q) in Ciao compiles the constraints in the program to a sequence of calls to the underlying constraints solver (at compile-time). This results in efficient implementation, since the structure of the constraints is processed only at compile-time, but requires the constraints to be known at static time which can be a limitation for metaprogramming-based applications such as static program analyzers. For example, the call:

```
?- X=(A+B), Y=(C-D), X .>. Y.

no
```

fails because `X .>. Y` is translated first to a sequence of calls that require (when they invoked) `X` and `Y` to be either numbers or free variables. To overcome this limitation, you can use `clp_meta/1` which delays the translation of the constraints from compile-time to run-time (i.e., when `clp_meta/1` is called), For example:

```
?- X=(A+B),Y=(C-D), clp_meta([X .>. Y]).

X = A+B,
Y = C-D,
C.<.D+A+B ?
```

The argument of `clp_meta/1` accepts a goal or lists of goals, where each goal is limited to conjunctions, disjunctions, or CLP(Q) constraints. Other operations on constraints which are extensively used in meta-programming, in particular in static program analysis, are *projection* and *entailment check*. The projection operation restricts the constraints (that are available in the store) to a given set of variables and turns the answer into terms. You can use the multifile predicate `dump_constraints/3` for that purpose:

```
?- A .>. C, C .>. B, dump_constraints([A,B],[X,Y],Cs).

Cs = [X.>.Y],
C.>.B,
C.<.A ?

?- C=(B+D), clp_meta([A .>. C, D .>. 0]), dump_constraints([A,B],[X,Y],Cs).

C = B+D,
Cs = [Y.<.X],
D.<. -B+A,
D.>.0 ?
```

The *entailment check* is used to check if a list of constrains is entailed by the store. You can use the predicate `clp_entailed/1` for that purpose:

```
?- A .>. C, C .>. B, B .>. D, clp_entailed([ A .>. B, A .>. D]).

B.>.D,
C.>.B,
C.<.A ?

yes

?- A .>=. B, clp_entailed([ A .>. B ]).

no
```

114.3 Known bugs and planned improvements (`clpq_doc`)

- clp(Q) and clp(R) cannot be used simultaneously in the same program, or even within the same toplevel session.

115 Constraint programming over reals

Author(s): Christian Holzbaur, Daniel Cabeza, Samir Genaim (Meta-programming predicates).

Note: This package is currently being adapted to the new characteristics of the Ciao module system. This new version now works right now to some extent, but it under further development at the moment. Use with (lots of) caution.

115.1 Usage and interface (clpr_doc)

- **Library usage:**
  `:- use_package(clpr).`
  or
  `:- module(...,...,[clpr]).`
- **New operators defined:**
  `.=./2` [700,xfx], `.<>./2` [700,xfx], `.<./2` [700,xfx], `.=<./2` [700,xfx], `.>./2` [700,xfx], `.>=./2` [700,xfx].

115.2 Other information (clpr_doc)

115.2.1 Some CLP(R) examples

(Other examples can be found in the source and library directories.)

- 'Reversible' Fibonacci (clpr):

```
:- module(_, [fib/2], []).
:- use_package(clpr).

fib(X,Y):- X .=. 0, Y .=. 0.
fib(X,Y):- X .=. 1, Y .=. 1.
fib(N,F) :-
        N .>. 1,
        N1 .=. N - 1,
        N2 .=. N - 2,
        fib(N1, F1),
        fib(N2, F2),
        F .=. F1+F2.
```

- Dirichlet problem for Laplace's equation (clpr):

```
%
% Solve the Dirichlet problem for Laplace's equation using
% Leibman's five-point finit-differenc approximation.
% The goal ?- go1 is a normal example, while the goal ?- go2
% shows output constraints for a small region where the boundary conditions
```

```
% are not specified.
%
:- use_package(clpq).
:- use_module(library(format)).

laplace([_, _]).
laplace([H1, H2, H3|T]):-
        laplace_vec(H1, H2, H3),
        laplace([H2, H3|T]).

laplace_vec([_, _], [_, _], [_, _]).
laplace_vec([_TL, T, TR|T1], [ML, M, MR|T2], [_BL, B, BR|T3]):-
        B + T + ML + MR - 4 * M .=. 0,
        laplace_vec([T, TR|T1], [M, MR|T2], [B, BR|T3]).

printmat([]).
printmat([H|T]):-
        printvec(H),
        printmat(T).

printvec([]):- nl.
printvec([H|T]):-
        printrat(H),
        printvec(T).

printrat(rat(N,D)) :- !,
        X is N/D,
        format(" ~2f",X).
printrat(N) :-
        X is N*100,
        format(" ~2d",X).

go1:-
        X =  [
      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, _, _, _, _, _, _, _, _, _, 100],
    [100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100]
    ],
        laplace(X),
        printmat(X).

% Answer:
% 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

```
% 100.00 51.11 32.52 24.56 21.11 20.12 21.11 24.56 32.52 51.11 100.00
% 100.00 71.91 54.41 44.63 39.74 38.26 39.74 44.63 54.41 71.91 100.00
% 100.00 82.12 68.59 59.80 54.97 53.44 54.97 59.80 68.59 82.12 100.00
% 100.00 87.97 78.03 71.00 66.90 65.56 66.90 71.00 78.03 87.97 100.00
% 100.00 91.71 84.58 79.28 76.07 75.00 76.07 79.28 84.58 91.71 100.00
% 100.00 94.30 89.29 85.47 83.10 82.30 83.10 85.47 89.29 94.30 100.00
% 100.00 96.20 92.82 90.20 88.56 88.00 88.56 90.20 92.82 96.20 100.00
% 100.00 97.67 95.59 93.96 92.93 92.58 92.93 93.96 95.59 97.67 100.00
% 100.00 98.89 97.90 97.12 96.63 96.46 96.63 97.12 97.90 98.89 100.00
% 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0 100.0

go2([B31, M32, M33, B34, B42, B43, B12, B13, B21, M22, M23, B24]) :-
        laplace([
            [_B11, B12, B13, _B14],
            [B21, M22, M23, B24],
            [B31, M32, M33, B34],
            [_B41, B42, B43, _B44]
    ]).


% Answer:
%
% B34.=. -4*M22+B12+B21+4*M33-B43,
% M23.=.4*M22-M32-B12-B21,
% B31.=. -M22+4*M32-M33-B42,
% B24.=.15*M22-4*M32-4*B12-4*B21-M33-B13 ?
```

115.2.2 Meta-programming with CLP(R)

see Section 114.2.2 [Meta-programming with CLP(Q)], page 598

115.3 Known bugs and planned improvements (`clpr_doc`)

- clp(Q) and clp(R) cannot be used simultaneously in the same program, or even within the same toplevel session.

116 Fuzzy Prolog

Author(s): Claudio Vaucheret, Sergio Guadarrama, Francisco Bueno.

This package impements an extension of prolog to deal with uncertainty. We implement a fuzzy prolog that models interval-valued fuzzy logic. This approach is more general than other fuzzy prologs in two aspects:

1. Truth values are sub-intervals on [0,1]. In fact, it could be a finite union of sub-intervals, as we will see below. Having a unique truth value is a particular case modeled with a unitary interval.
2. Truth values are propagated through the rules by means of a set of *aggregation operators*. The definition of an *aggregation operator* is a generalization that subsumes conjunctive operators (triangular norms as min, prod, etc.), disjunctive operators (triangular co-norms as max, sum, etc.), average operators (averages as arithmetic average, cuasi-linear average, etc.) and hybrid operators (combinations of previous operators).

We add uncertainty using CLP(R) instead of implementing a new fuzzy resolution as other fuzzy prologs. In this way, we use the original inference mechanism of Prolog, and we use the constraints and its operations provided by CLP(R) to handle the concept of partial truth. We represent intervals as constrains over real numbers and *aggregation operators* as operations with constraints.

Each fuzzy predicate has an additional argument which represents its truth value. We use ":~" instead of ":-" to distinguish fuzzy clauses from prolog clauses. In fuzzy clauses, truth values are obtained via an aggregation operator. There is also some syntactic sugar for defining fuzzy predicates with certain membership functions, the fuzzy counterparts of crisp predicates, and the fuzzy negation of a fuzzy predicate.

116.1 Usage and interface (`fuzzy_doc`)

- **Library usage:**
  `:- use_package(fuzzy).`
  or
  `:- module(...,...,[fuzzy]).`
- **Exports:**
  - *Predicates:*
    `:#/2`, `fuzzy_predicate/1`, `fuzzy/1`, `fnot/1`, `:~/2`, `=>/4`.
  - *Properties:*
    `fuzzybody/1`.
  - *Regular Types:*
    `faggregator/1`.
- **New operators defined:**
  `:~/2` [1200,xfx], `:~/1` [1200,xf], `:=/2` [1200,xfx], `:=/1` [1200,xf], `:#/2` [1200,xfx], `=>/1` [1175,fx], `fnot/1` [1150,fx], `aggr/1` [1150,fx], `##/2` [1120,xfy], `<#/2` [1120,xfy], `#>/2` [1120,xfy], `fuzzy/1` [1150,fx], `fuzzy_predicate/1` [1190,fx], `fuzzy_discrete/1` [1190,fx].
- **New declarations defined:**
  `aggr/1`.

116.2 Documentation on new declarations (fuzzy_doc)

aggr/1: DECLARATION

Usage: :- aggr(Name).

– *Description:* Declares Name an aggregator. Its binary definition has to be provided. For example:

```
:- aggr myaggr.

myaggr(X,Y,Z):- Z .=. X*Y.
```

defines an aggregator identical to prod.

– *The following properties hold at call time:*

Name is an atomic term (an atom or a number). (basic_props:constant/1)

116.3 Documentation on exports (fuzzy_doc)

:#/2: PREDICATE

Usage: :#(Name,Decl)

– *Description:* Defines fuzzy predicate Name from the declaration Decl.

– *The following properties hold upon exit:*

Name is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

Decl is one of the following three:

```
fuzzydecl(fuzzy_predicate(_1)).
fuzzydecl(fuzzy(_1)).
fuzzydecl(fnot(_1)).
```

(user(/home/ciaotester/tests/auto/CiaoDE/ciao/library/fuzzy/fuzzy_doc):fuzzydecl/1)

fuzzy_predicate/1: PREDICATE

Usage: fuzzy_predicate(Domain)

– *Description:* Defines a fuzzy predicate with piecewise linear continuous membership function. This is given by Domain, which is a list of pairs of domain-truth values, in increasing order and exhaustive. For example:

```
young :# fuzzy_predicate([(0,1),(35,1),(45,0),(120,0)]).
```

defines the predicate:

```
young(X,1):- X .>=. 0, X .<. 35.
young(X,M):- X .>=. 35, X .<. 45, 10*M .=. 45-X.
young(X,0):- X .>=. 45, X .=<. 120.
```

– *The following properties should hold at call time:*

Domain is a list. (basic_props:list/1)

fuzzy/1: PREDICATE

Usage: `fuzzy(Name)`

– *Description:* Defines a fuzzy predicate as the fuzzy counterpart of a crisp predicate `Name`. For example,

```
p_f :# fuzzy p/2
```

defines a new fuzzy predicate `p_f/3` (the last argument is the truth value) with truth value equal to 0 if `p/2` fails and 1 otherwise.

– *The following properties should hold at call time:*

`Name` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

fnot/1: PREDICATE

Usage: `fnot(Name)`

– *Description:* Defines a fuzzy predicate as the fuzzy negation of another fuzzy predicate `Name`. For example,

```
notp_f :# fnot p_f/3
```

defines the predicate:

```
notp_f(X,Y,M) :-
        p_f(X,Y,Mp),
        M .=. 1 - Mp.
```

– *The following properties should hold at call time:*

`Name` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)

:~/2: PREDICATE

Usage: `:~(Head,Body)`

– *Description:* Defines a fuzzy clause for a fuzzy predicate. The clause contains calls to either fuzzy or crisp predicates. Calls to crisp predicates are automatically fuzzified. The last argument of `Head` is the truth value of the clause, which is obtained as the aggregation of the truth values of the body goals. An example:

```
:- module(young2,_,[fuzzy]).

young_couple(X,Y,Mu) :~ min
        age(X,X1),
        age(Y,Y1),
        young(X1,MuX),
        young(Y1,MuY).

age(john,37).
```

```
age(rose,39).

young :# fuzzy_predicate([(0,1),(35,1),(45,0),(120,0)]).
```

so that:

```
?- young_couple(john,rose,M).

M .=. 0.6 ?
```

– *The following properties should hold at call time:*

`Head` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Body` is a clause body plus an optional aggregation operator. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/library/fuzzy/fuzzy_doc):fuzzybody/1)

fuzzybody/1: PROPERTY

A clause body, optionally prefixed by the name of an aggregation operator. The agregators currently provided are listed under `faggregator/1`. By default, the aggregator used is `min`.

Usage: `fuzzybody(B)`

– *Description:* `B` is a clause body plus an optional aggregation operator.

faggregator/1: REGTYPE

The first three are, respectively, the T-norms: minimum, product, and Lukasiewicz's. The last three are their corresponding T-conorms. Aggregators can be defined by the user, see `aggr/1`.

```
faggregator(min).
faggregator(prod).
faggregator(luka).
faggregator(max).
faggregator(dprod).
faggregator(dluka).
```

Usage: `faggregator(Aggr)`

– *Description:* `Aggr` is an aggregator which is cumulative, i.e., its application to several values by iterating pairwise the binary operation is safe.

=>/4: PREDICATE

Usage: `=>(Aggr,A,B,Truth)`

– *Description:* The fuzzy implication `A => B` defined by aggregator `Aggr`, resulting in the truth value `Truth`.

– *The following properties should hold at call time:*

`Aggr` is an aggregator which is cumulative, i.e., its application to several values by iterating pairwise the binary operation is safe. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/library/fuzzy/fuzzy_doc):faggregator/1)

`A` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`B` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`Truth` is a free variable. (term_typing:var/1)

116.4 Other information (`fuzzy_doc`)

An example program:

```
:- module(dicesum5,_,[fuzzy]).

% this example tries to measure which is the possibility
% that a couple of values, obtained throwing two loaded dice, sum 5. Let
% us suppose we only know that one die is loaded to obtain a small value
% and the other is loaded to obtain a large value.
%
% the query is  ? sum(5,M)
%

small :# fuzzy_predicate([(1,1),(2,1),(3,0.7),(4,0.3),(5,0),(6,0)]).
large :# fuzzy_predicate([(1,0),(2,0),(3,0.3),(4,0.7),(5,1),(6,1)]).

die1(X,M) :~
        small(X,M).

die2(X,M) :~
        large(X,M).


two_dice(X,Y,M):~ prod
        die1(X,M1),
        die2(Y,M2).

sum(2,M) :~
        two_dice(1,1,M1).


sum(5,M) :~ dprod
        two_dice(4,1,M1),
        two_dice(1,4,M2),
        two_dice(3,2,M3),
        two_dice(2,3,M4).
```

There are more examples in the subdirectory `fuzzy/examples` of the distribution.

116.5 Known bugs and planned improvements (`fuzzy_doc`)

- General aggregations defined by users.
- Inconsistent behaviour of meta-calls in fuzzy clauses.
- Some meta-predicate constructions need be added, specially for 'disjunctive' fuzzy clauses, e.g., `sum/2` in the dice example.

117 Object Oriented Programming

Author(s): Angel Fernandez Pineda.

O'Ciao is a set of libraries which allows object-oriented programming in Ciao Prolog. It extends the Ciao Prolog module system by introducing two new concepts:

- Inheritance.
- Instantiation.

Polymorphism is the third fundamental concept provided by object oriented programming. This concept is not mentioned here since **traditional PROLOG systems are polymorphic by nature**.

Classes are declared in the same way as modules. However, they may be enriched with inheritance declarations and other object-oriented constructs. For an overview of the fundamentals of O'Ciao, see `http://www.clip.dia.fi.upm.es/~clip/papers/ociao-tr.ps.gz`. However, we will introduce the concepts in a tutorial way via examples.

117.1 Early examples

The following one is a very simple example which declares a class – a simple stack. Note that if you replace *class/1* declaration with a *module/1* declaration, it will compile correctly, and can be used as a normal Prolog module.

```
%%------------------------------------------%%
%% A class for stacks.                      %%
%%------------------------------------------%%

%% Class declaration: the current source defines a class.
:- class(stack,[],[]).

% State declaration: storage/1 is an attribute.
:- dynamic storage/1.

% Interface declaration: the following predicates will
% be available at run-time.
:- export(push/1).
:- export(pop/1).
:- export(top/1).
:- export(is_empty/0).

% Methods

push(Item) :-
        nonvar(Item),
        asserta_fact(storage(Item)).

pop(Item) :-
        var(Item),
        retract_fact(storage(Item)).

top(Top) :-
        storage(Top), !.
```

```
is_empty :-
        storage(_), !, fail.
is_empty.
```

If we load this code at the Ciao toplevel shell:

```
?- use_package(objects).

yes
?- use_class(library('class/examples/stack')).

yes
?-
```

we can create two stack *instances* :

```
?- St1 new stack,St2 new stack.

St1 = stack('9254074093385163'),
St2 = stack('9254074091') ? ,
```

and then, we can operate on them separately:

```
1 ?- St1:push(8),St2:push(9).

St1 = stack('9254074093385163'),
St2 = stack('9254074091') ?

yes
1 ?- St1:top(I),St2:top(K).

I = 8,
K = 9,
St1 = stack('9254074093385163'),
St2 = stack('9254074091') ?

yes
1 ?-
```

The interesting point is that there are two stacks. If the previous example had been a normal module, we would have a stack , but **only one** stack.

The next example introduces the concepts of *inheritable* predicate, *constructor*, *destructor* and *virtual method*. Refer to the following sections for further explanation.

```
%%------------------------------------------%%
%% A generic class for item storage.        %%
%%------------------------------------------%%
:- class(generic).

% Public interface declaration:
:- export([set/1,get/1,callme/0]).

% An attribute
:- data datum/1.

% Inheritance declaration: datum/1 will be available to
% descendant classes (if any).
```

```
:- inheritable(datum/1).

% Attribute initialization: attributes are easily initialized
% by writing clauses for them.
datum(none).

% Methods

set(X) :-
        type_check(X),
        set_fact(datum(X)).

get(X) :-
        datum(X).

callme :-
        a_virtual(IMPL),
        display(IMPL),
        display(' implementation of a_virtual/0 '),
        nl.

% Constructor: in this case, every time an instance
% of this class is created, it will display a message.
generic :-
        display(' generic class constructor '),
        nl.

% Destructor: analogous to the previous constructor,
% it will display a message every time an instance
% of this class is eliminated.
destructor :-
        display(' generic class destructor '),
        nl.

% Predicates:
% cannot be called as messages (X:method)

% Virtual declaration: tells the system to use the most
% descendant implementation of a_virtual/1 when calling
% it from inside this code (see callme/0).
% If there is no descendant implementation for it,
% the one defined bellow will be used.
:- virtual a_virtual/1.

a_virtual(generic).

:- virtual type_check/1.

type_check(X) :-
        nonvar(X).
```

And the following example, is an extension of previous class. This is performed by establishing an inheritance relationship:

```
%%-----------------------------------------------%%
%% This class provides additional functionality %%
%% to the "generic" class.                       %%
%%-----------------------------------------------%%
:- class(specific).

% Establish an inheritance relationship with class "generic".
:- inherit_class(library('class/examples/generic')).

% Override inherited datum/1.
% datum/1 is said to be overriden because there are both an
% inherited definition (from class "generic") and a local one,
% which overrides the one inherited.
:- data datum/1.
:- inheritable datum/1.

% Extend the public interface inherited from "generic".
% note that set/1 and a_virtual/0 are also overriden.
% undo/0 is a new functionality added.
:- export([set/1,undo/0]).

% Methods

set(Value) :-
        inherited datum(OldValue),
        !,
        inherited set(Value),
        asserta_fact(datum(OldValue)).
set(Value) :-
        inherited set(Value).

undo :-
        retract_fact(datum(Last)), !,
        asserta_fact(inherited(datum(Last))).
undo :-
        retractall_fact(inherited(datum(_))).

% Constructor
specific :-
        generic,
        retractall_fact(inherited(datum(_))),
        display(' specific class constructor '),
        nl.

% Destructor
destructor :-
        display(' specific class destructor '),
        nl.
```

```
% Predicates

% New implementation of a_virtual/1.
% Since this predicate was declared virtual, the
% implementation below will be called from the inherited
% method callme/0 instead of the version defined at "generic".
a_virtual(specific).
```

Additional examples may be found on the *library/class/examples* directory relative to your Ciao Prolog instalation.

117.2 Recommendations on when to use objects

We would like to give some advice in the use of object oriented programming, in conjunction with the declarative paradigm.

You should reconsider using O'Ciao in the following cases:

- The pretended "objects" have no state,i.e., no data or dynamic predicates. In this case, a normal module will suffice.
- There is state, but there will be only one instance of a pretended class. Again, a module suffices.
- The "objects" are data structures (list,trees,etc) already supported by Prolog. However, it does make sense to model, using objects, data structures whose change implies a side-effect such as drawing a particular window on the screen.

We recommend the usage of O'Ciao in the following cases:

- You feel you will need to have several copies of a "module".
- Local copies of a module are needed instead of a global module beeing modified by several ones.
- The "classes" are a representation of external entities to Prolog. For example: the X-Window system.
- There is state or code outside the Prolog system which needs to be manipulated. For example: interfaces to Java or Tcl/Tk code.
- You are not familiar with Prolog, but you know about object oriented programming. O'Ciao may be used as a learning tool to introduce yourself on the declarative programming paradigm.

117.3 Limitations on object usage

O'Ciao run-time speed is limited by the usage of meta-programming structures, for instance: `X = (Object:mymethod(25)), call(X)`. O'Ciao will optimize static manipulation of objects (those that can be determined at compile time).

118 Declaring classes and interfaces

Author(s): Angel Fernandez Pineda.

O'Ciao classes are declared in the same way as traditional prolog modules. The general mechanism of *source expansion* will translate object-oriented declarations to normal prolog code. This is done transparently to the user.

Abstract *interfaces* are restricted classes which declare exported predicates with no implementation. The implementation itselt will be provided by some class using an `implements/1` declaration. Only `export/1` and `data/1` declarations are allowed when declaring an interface. Normal classes may treated as interfaces just ignoring all exported predicate implementations.

118.1 Usage and interface (`class_doc`)

- **Library usage:**

  To declare a class the compiler must be told to use the `class` *source expansion*. To do so, source code must start with a module declaration which loads the class package:

  ```
  :- class(ClassName).
  ```

  or a `module/3` declaration, as follows:

  ```
  :- module(ClassName,[],[class]).
  ```

  interfaces are declared in a similar way:

  ```
  :- interface(InterfaceName).
  ```

  Please, do not use SICStus-like module declaration, with a non-empty export list. In other case, some non-sense errors will be reported by normal Ciao module system.

  Most of the regular Ciao declarations may be used when defining a class, such as `concurrent/1`, `dynamic/1`, `discontiguous/1`, `multifile/1`, and so on.

  However, there are some restrictions wich apply to those declarations:

  - `meta_predicate/1` declaration is not allowed to hold *addmodule and pred(N) meta-arguments*, except for previously declared multifiles.
  - Attribute and multifile predicates must be declared before any clause of the related predicate.
  - There is no sense in declaring an attribute as meta_predicate.

  It is a good practique to put all your declarations at the very begining of the file, just before the code itself.
- **Exports:**
  - *Predicates:*

    `inherited/1`, `self/1`, `constructor/0`, `destructor/0`.
- **New declarations defined:**

  `export/1`, `public/1`, `inheritable/1`, `data/1`, `dynamic/1`, `concurrent/1`, `inherit_class/1`, `implements/1`, `virtual/1`.
- **Other modules used:**
  - *System library modules:*

    `objects/objects_rt`.

118.2 Documentation on new declarations (`class_doc`)

export/1: DECLARATION

Declares a method or attribute to be part of the *public interface*.

The public interface is the set of predicates wich will be accesible from any code establishing an usage relationship with this class (see `use_class/1` for further information).

Publishing an attribute or method is very similar to *exporting* a predicate in a Prolog module.

Whether an inherited and exported predicate is overriden, it must be explicitly exported again.

An inherited (but not exported) predicate may become exported, without overriding it by the usage of this declaration.

Usage: :- `export(Spec)`.

- *Description:* `Spec` will be part of the public (exported) interface.
- *The following properties should hold at call time:*

  `Spec` is a method or attribute specification. (objects_rt:method_spec/1)

public/1: DECLARATION

Just an alias for `export/1`.

Usage: :- `public(Spec)`.

- *Description:* This declaration may be used instead of `export/1`.
- *The following properties should hold at call time:*

  `Spec` is a method or attribute specification. (objects_rt:method_spec/1)

inheritable/1: DECLARATION

Declares a method or attribute to be inherited by descendant classes. Notice that all **public predicates are inheritable by default**. There is no need to mark them as inheritable.

Traditionaly, object oriented languages makes use of the *protected* concept. Inheritable/1 may be used as the same concept.

The set of inheritable predicates is called the *inheritable interface*.

Usage: :- `inheritable(MethodSpec)`.

- *Description:* `MethodSpec` is accessible to descendant classes.
- *The following properties should hold at call time:*

  `MethodSpec` is a method or attribute specification. (objects_rt:method_spec/1)

data/1: DECLARATION

Declares an *attribute* at current class. Attributes are used to build the internal state of instances. So, each instance will own a particular copy of those attribute definitions. In this way, one instance may have different state from another.

O'Ciao attributes are restricted to hold simple facts. It is not possible to hold a Head :- Body clause at an instance attribute.

Notice that attributes are *multi-evaluated* by nature, and may be manipulated by the habitual **assert/retract** family of predicates.

Attributes may also be initialized. In order to do so, simply put some clauses after the attribute definition. Each time an instance is created, its initial state will be built from those *initialization clauses*.

Note: whether a data/1 declaration appears inside an interface, it will be automatically exported.

Usage: :- `data Spec`.

- *Description:* `Spec` is an attribute.
- *The following properties should hold at call time:*

  `Spec` is a method or attribute specification. (objects_rt:method_spec/1)

dynamic/1: DECLARATION

Just an alias for `data/1`.

Usage: :- `dynamic Spec`.

- *Description:* You may use this declaration instead of `data/1`.
- *The following properties should hold at call time:*

  `Spec` is a method or attribute specification. (objects_rt:method_spec/1)

concurrent/1: DECLARATION

Declares a *concurrent attribute* at current class. Concurrent attributes are just the same as normal attributes, those declared using `data/1`, except for they may freeze the calling thread instead of failing when no more choice points are remaining on the concurrent attribute.

In order to get more information about concurrent behavior take a look to the concurrent/1 built-in declaration on Ciao Prolog module system.

Usage: :- `concurrent Spec`.

- *Description:* Declares `Spec` to be a concurrent attribute.
- *The following properties should hold at call time:*

  `Spec` is a method or attribute specification. (objects_rt:method_spec/1)

inherit_class/1: DECLARATION

Makes any public and/or inheritable predicate at inherited class to become accesible by any instance derived from current class.

Inherited class is also called the *super class*.

Only one inherit_class/1 declaration is allowed to be present at current source.

Notice that inheritance is public by default. Any public and/or inheritable declaration will remain the same to descendant classes. However, any inherited predicate may be *overriden* (redefined).

A predicate is said to be *overriden* when it has been inherited from super class, but there are clauses (or a `data/1` declaration) present at current class for such a predicate.

Whether a **public** predicate is overriden, the local definition must also be exported, otherwise an error is reported.

Whether an **inheritable** predicate (not public) is overriden, the local definition must also be marked as inheritable or exported, otherwise an error is also reported.

Note: whether inherit_class/1 appears inside an interface, it will be used as an `implements/1` declaration.

Usage: :- `inherit_class(Source)`.

– *Description:* Establish an *inheritance relationship* between current class and the class defined at `Source` file.
– *The following properties should hold at call time:*

`Source` is a valid path to a prolog file containing a class declaration (without .pl extension). (objects_rt:class_source/1)

implements/1: DECLARATION

Forces current source to provide an implementation for the given interface file. Such interface file may declare another class or a specific interface.

Every public predicate present at given interface file will be automatically declared as public at current source, so you **must** provide an implementation for such predicates.

The effect of this declaration is called *interface inheritance*,and there is no restriction on the number of implements/1 declarations present at current code.

Usage: :- `implements(Interface)`.

– *Description:* Current source is supposed to provide an implementation for `Interface`.
– *The following properties should hold at call time:*

`Interface` is a valid path to a prolog file containing a class declaration or an interface declaration (without .pl extension). (objects_rt:interface_source/1)

virtual/1: DECLARATION

This declaration may be used whenever descendant classes are to implement different versions of a given predicate.

virtual predicates give a chance to handle, in an uniform way, different implementations of the same functionality.

Whether a virtual predicate is declared as a method, there must be at least one clause of it present at current source. Whenever no special implementation is needed at current class, a never-fail/allways-fail clause may be defined (depending on your needs). For example:

```
:- virtual([ test1/1 , test2/2 ]).
test1(_).
test2(_,_) :- fail.
```

This kind of virtual methods are also known as *abstract methods*, since implementation is fully delegated to descendant classes.

An attribute may be also declared as a virtual one, but there is no need to write clauses for it.

Usage: :- `virtual(VirtualMethodSpec)`.

– *Description:* All calls to `VirtualMethodSpec` predicate in current source will use the most descendant implementation of it.
– *The following properties should hold at call time:*

`VirtualMethodSpec` is a method specification. (objects_rt:virtual_method_spec/1)

118.3 Documentation on exports (`class_doc`)

inherited/1: PREDICATE

This predicate qualificator may be used whenever you need to reference an attribute or method on the super class.

Since methods and attributes may be redefined, this qualificator is need to distinguish between a locally declared predicate and the inherited one, which has the same name.

There is no need to use inherited/1 if a particular inherited predicate has not been redefined at current class.

Usage: `inherited(Goal)`

- *Description:* References a given `Goal` at the super class
- *The following properties should hold at call time:*

  `Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

self/1: PREDICATE

Determines which instance is currently executing self/1 goal.

Predicate will fail if argument is not a free variable. Otherwise, it will allways succeed, retrieving the instance identifier which is executing current code.

This functionality is very usefull since an object must have knowledge of other object's identifier in order to send messages to it.For example:

:- concurrent ack/0.

send_data_to_object(Data,Obj) :- self(X), Obj:take_this(Data,X), current_fact(ack).

acknowledge :- asserta_fact(ack).

take_this(Data,Sender) :- validate_data(Data), Sender:acknowledge.

Usage: `self(Variable)`

- *Description:* Retrieves current instance identifier in `Variable`
- *The following properties should hold at call time:*

  `Variable` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `Variable` is an unique term which identifies an object. (objects_rt:instance_id/1)

constructor/0: PREDICATE

A *constructor* is a special case of method which complains the following conditions:

- The constructor functor matches the current class name.
- A constructor may hold any number of arguments.
- If an inheritance relationship was defined, an inherited constructor must be manually called (see below).
- When instance creation takes place, any of the declared constructors are implicitly called. The actual constructor called depends on the `new/2` goal specified by the user.

This is a simple example of constructor declaration for the foo class:

```
foo :-
    display('an instance was born').
```

Constructor declaration is not mandatory, and there may be more than one constructor declarations (with different arity) at the source code.

This functionality is usefull when some computation is needed at instance creation. For example: opening a socket, clearing the screen, etc.

Whenever an inheritance relationship is established, and there is any constructor defined at the super class, you must call manually an inherited constructor. Here is an example:

```
:- class(foo).
:- inherit_class(myclass).

foo :-
    myclass(0),
    display('an instance was born').

foo(N) :- myclass(N).
```

Consequences may be unpredictable, if you forget to call an inherited constructor. You should also take care not to call an inherited constructor twice.

All defined constructors are inheritable by default. A constructor may also be declared as public (by the user), but it is not mandatory.

Usage:

– *Description:* Constructors are implicitly declared

destructor/0: PREDICATE

A *destructor* is a special case of method which will be automatically called when instance destruction takes place.

A destructor will never be wanted to be part of the public interface, and there is no need to mark them as inheritable, since all inherited destructors are called by O'Ciao just before yours.

This is a simple example of destructor declaration:

```
destructor :-
    display('goodbye, cruel world!!!').
```

Destructor declaration is not mandatory. Failure or sucess of destructors will be ignored by O'Ciao, and they will be called only once.

This functionality is useful when some computation is need at instance destruction. For example: closing an open file.

Usage:

– *Description:* Destructors are implicitly declared

118.4 Other information (class_doc)

This describes the errors reported when declaring a class or an interface. The first section will explain compile-time errors, this is, any semantic error which may be determined at compile time. The second section will explain run-time errors, this is, any exception that may be raisen by the incorrect usage of O'Ciao. Some of those errors may be not reported at compile time, due to the use of meta-programational structures. For example:

```
functor(X,my_method,0),call(X).
```

O'Ciao is not able to check whether my_method/0 is a valid method or not. So, this kind of checking is left to run time.

118.4.1 Class and Interface error reporting at compile time

- **ERROR : multiple inheritance not allowed.**

  There are two or more inherit_class/1 declarations found at your code. Only one declaration is allowed, since there is no multiple code inheritance support.
- **ERROR : invalid inheritance declaration.**

  The given parameter to inherit_class/1 declaration is not a valid path to a Prolog source.
- **ERROR : sorry, addmodule meta-arg is not allowed at *F/A*.**

  You are trying to declare *F/A* as meta-predicate, and one of the meta-arguments is *addmodule*. This is not allowed in O'Ciao due to implementation restrictions. For example:

  ```
  :- meta_predicate example(addmodule).

  example(X,FromModule) :- call(FromModule:X).
  ```
- **ERROR : invalid attribute declaration for *Arg*.**

  Argument to data/1 or dynamic/1 declaration is not a valid predicate specification of the form *Functor/Arity*. For example:

  ```
  :- data attr.

  :- dynamic attr(_).

  :- data attr/m.
  ```

  etc,etc...
- **ERROR : pretended attribute *F/A* was assumed to be a method.**

  You put some clauses of *F/A* before the corresponding data/1 or dynamic/1 declaration. For example:

  ```
  attr(initial_value).

  :- data attr/1.
  ```

  It is a must to declare attributes before any clause of the given predicate.
- **ERROR : destructor/0 is not allowed to be an attribute.**

  There is a :- data(destructor/0) or :- dynamic(destructor/0). declaration in your code. This is not allowed since destructor/0 is a reserved predicate, and must be allways a method.
- **ERROR : *Constructor* is not allowed to be an attribute.**

  As the previos error, you are trying to declare a constructor as an attribute. A constructor must be allways a method.
- **ERROR : invalid multifile: destructor/0 is a reserved predicate.**

  There is a :- multifile(destructor/0). declaration in your code. This is not allowed since destructor/0 is a reserved predicate, and must be allways a method.
- **ERROR : invalid multifile: *Constructor* is a reserved predicate.**

  As the previos error, you are trying to declare a constructor as a multifile. Any constructor must allways be a method.
- **ERROR : multifile declaration of *F/A* ignored: it was assumed to be a method.**

  You put some clauses of *F/A* before the corresponding multifile/1 declaration. For example:

  example(a,b).

  ```
  :- multifile example/2.
  ```

  Multifile predicates must be declared before any clause of the given predicate.
- **ERROR : invalid multifile declaration: multifile(*Arg*).**

  Given argument to multifile/1 declaration is not a valid predicate specification, of the form *Functor/Arity*.

- **ERROR : invalid public declaration:** *Arg*.
  Given argument *Arg* to public/1 or export/1 declaration is not a valid predicate specification, of the form *Functor/Arity*.
- **ERROR : invalid inheritable declaration: inheritable(***Arg***).**
  Given argument *Arg* to inheritable/1 declaration is not a valid predicate specification, of the form *Functor/Arity*.
- **ERROR : destructor/0 is not allowed to be virtual.**
  There is a :- virtual(destructor/0) declaration present at your code. Destructors and/or constructors are not allowed to be virtual.
- **ERROR :** *Constructor* **is not allowed to be virtual.**
  As the previous error, you are trying to declare a constructor as virtual. This is not allowed.
- **ERROR : invalid virtual declaration: virtual(***Arg***).**
  Given argument to virtual/1 declaration is not a valid predicate specification, of the form *Functor/Arity*.
- **ERROR : clause of** *F/A* **ignored : only facts are allowed as initial state.**
  You declared *F/A* as an attribute, then you put some clauses of that predicate in the form *Head :- Body*. For example:
  ```
  :- data my_attribute/1.
  my_attribute(X) :- X>=0 , X<=2.
  ```
  This is not allowed since attributes are assumed to hold simple facts. The correct usage for those *initialization clauses* is:
  ```
  :- data my_attribute/1.
  my_attribute(0).
  my_attribute(1).
  my_attribute(2).
  ```
- **ERROR : multifile** *F/A* **is not allowed to be public.**
  The given *F/A* predicate is both present at multifile/1 and public/1 declarations. For example:
  ```
  :- public(p/1).
  :- multifile(p/1).
  ```
  This is not allowed since multifile predicates are not related to Object Oriented Programming.
- **ERROR : multifile** *F/A* **is not allowed to be inheritable.**
  Analogous to previous error.
- **ERROR : multifile** *F/A* **is not allowed to be virtual.**
  Analogous to previous error.
- **ERROR : virtual** *F/A* **must be a method or attribute defined at this class.**
  There is a virtual/1 declaration for *F/A*, but there is not any clause of that predicate nor a `data/1` declaration. You must declare at least one clause for every virtual method. Virtual attributes does not require any clause but a `data/1` declaration must be present.
- **ERROR : implemented interface** *Module* **is not a valid interface.**
  There is an `implements/1` declaration present at your code where given *Module* is not declared as class nor interface.
- **ERROR : predicate** *F/A* **is required both as method (at** *Itf1* **interface) and attribute (at** *Itf2* **interface).**
  There is no chance to give a correct implementation for *F/A* predicate since *Itf1* and *Itf2* interfaces require different definitions. To avoid this error, you must remove one of the related `implements/1` declaration.

- **ERROR : inherited** *Source* **must be a class.**

  There is an :- inherit_class(*Source*) declaration, but that source was not declared as a class.
- **ERROR : circular inheritance:** *Source* **is not a valid super-class.**

  Establishing an inheritance relationship with *Source* will cause current class to be present twice in the inheritance line. This is not allowed. The cause of this is error is simple : There is some inherited class from *Source* which also establishes an inheritance relationship with current source.
- **ERROR : method/attribute** *F/A* **must be implemented.**

  Some of the implemented interfaces requires *F/A* to be defined, but there is no definition for such predicate, even an inherited one.
- **ERROR : local implementation of** *F/A* **hides inheritable/public definition.**

  There is an inherited definition for *F/A* which is been redefined at current class, but there is no valid inheritable/public declaration for the last one. Overriden public predicates must be also declared as public. Overriden inheritable predicates must be declared either as public or inheritable.
- **ERROR : public predicate** *F/A* **was not defined nor inherited.**

  There is a `public/1` declaration for *F/A*, but there is no definition for it at current class nor an inherited one.
- **ERROR : argument to self/1 must be a free variable.**

  Argument to self/1 is not a variable, for example: `self(abc)`.
- **ERROR : unknown inherited attribute in** *Goal***.**

  Goal belongs to assert/retract family of predicates, and given argument is not a valid inherited attribute. The most probable causes of this error are:
  - The given predicate is defined at super-class, but you forgot to mark it as inheritable (or public), at such class.
  - The given predicate was not defined (at super-class) as an attribute, just as a method.
- **ERROR : unknown inherited goal:** *Goal***.**

  The given *Goal* was not found at super-class, or it is not accessible. Check whether *Goal* was marked as inheritable (or public) at super-class.
- **ERROR : invalid argument:** *F/A* **is not an attribute.**

  You are trying to pass a method as an argument to any predicate which expect a *fact* predicate.
- **ERROR : unknown inherited fact:** *Fact***.**

  There is a call to any predicate which expects a *fact* argument (those declared as data or dynamic),but the actual argument is not an inherited attribute.For example:

  ```
  asserta_fact(inherited(not_an_attribute(8)))
  ```

  where not_an_attribute/1 was not declared as data or dynamic by the super-class (or corresponding ascendant).
- **ERROR : unknown inherited spec:** *F/A***.**

  There is a reference to an inherited predicate specification, but the involved predicate has not been inherited.
- **WARNING : meta-predicate specification of** *F/A* **ignored since this is an attribute.**

  You declared *F/A* both as an attribute and a meta-predicate. For example:

  ```
  :- meta_predicate attr(goal).
  :- data attr/1.
  ```

  There is no sense in declaring an attribute as meta-predicate.

- **WARNING : class destructor is public**

  There is a :- public(destructor/0) declaration present at your code. Marking a destructor as public is a very bad idea since anybody may destroy or corrupt an instance before the proper time.
- **WARNING : class destructor is inheritable**

  Analogous to previous error.
- **WARNING : There is no call to inherited constructor/s**

  You have not declared any constructor at your class, but there is any inherited constructor that should be called. Whenever you do not need constructors, but there is an inheritance relationship (where super-class declares a constructor), you should write a simple constructor as the following example:

```
:- class(myclass).
:- inherit_class(other_class).

myclass :-
        other_class.
```

- **WARNING : multifile *F/A* hides inherited predicate.**

  You declared as multifile a predicate which matches an inherited predicate name. Any reference to the inherited predicate must be done by the ways of the inherited/1 qualificator.

118.4.2 Class and Interface error reporting at run time

- **EXCEPTION : error(existence_error(object_goal,*Goal*),*Mod*).**

  Called *Goal* from module (or class) *Mod* is unknown or has not been published.

118.4.3 Normal Prolog module system interaction

O'Ciao works in conjunction with the Ciao Prolog module system, which also reports its own error messages. This will cause Ciao to report a little criptic error messages due to the general mechanism of source-to-source expansion. Those are some tips you must consider when compiling a class:

- Any error relative to method 'm' with arity A will be reported for predicate 'obj$m'/A+1. For example :

  `WARNING: (lns 28-30) [Item,Itema] - singleton variables in obj$remove/2`

  This error is relative to method remove/1.
- `set_prolog_flag/1` declaration will be usefull when declaring multiple constructors. It will avoid some awful warnings. Example:

```
:- class(myclass).

%% Use this declaration whenever several constructors are needed.

:- set_prolog_flag(multi_arity_warnings,off).

myclass(_).

myclass(_,_).

:- set_prolog_flag(multi_arity_warnings,on).
```

118.5 Known bugs and planned improvements (`class_doc`)

- addmodule and pred(N) meta-arguments are not allowed on meta-predicates.

119 Compile-time usage of objects

Author(s): Angel Fernandez Pineda.

This package is required to enable user code to create objects and manipulate them, as well as loading any needed class.

119.1 Usage and interface (objects_doc)

- **Library usage:**
  Any code which needs to use objects must include the objects package:
  `:- module(`*ModuleName*`,`*Exports*`,[objects]).`
  You can use objects even if your code is a class. Note that declaring a class does not automatically enables the code to create instances.
  `:- class(`*ModuleName*`,[],[objects]).`
  This package enables both static and dynamic usage of objects.
- **New declarations defined:**
  `use_class/1`, `instance_of/2`, `new/2`.
- **Other modules used:**
  - *System library modules:*
    `objects/objects_rt`.

119.2 Documentation on new declarations (objects_doc)

use_class/1: DECLARATION

It establishes an usage relationship between the given file (which is supposed to declare a class) and current source. Usage relationships are needed in order to enable code to create instances of the given class, and to make calls to instances derived from such class.

Since an interface is some kind of class, they may be used within this declaration but only for semantic checking porpouses. Instances will not be derived from interfaces.

use_class/1 is used in the same way as `use_module/1`.

Usage: `:- use_class(ClassSource).`

- *Description:* Establish usage relationship with `ClassSource`.
- *The following properties should hold at call time:*
  `ClassSource` is a valid path to a prolog file containing a class declaration (without .pl extension). (objects_rt:class_source/1)

instance_of/2: DECLARATION

Statically declares an identifier to be an instance of a given class.

It may be used as `new/2` predicate except for:

- The instance identifier will not be a variable, it must be provided by the user, and must be unique.
- Instance creation will never fail, even if the constructor fails.

For every statically declared object the given constructor will be called at program startup. Those instances may be destroyed manually, but it is not recommended.

When reloading the involved class from the Ciao toplevel shell. It may destroy statically declared instances, and create them again.

Statically declared instances must be called using a specifically designed module-qualification: `ClassName(Object):Goal`. For example:

```
:- module(example,[main/0],[objects]).
:- use_class(library(counter)).
:- cnt instance_of counter(10).

main :-
     counter(cnt):decrease(1),
     counter(cnt):current_value(X),
     display(X).
```

But **statically written code** (only) is allowed to use module-style qualifications as a macro:

```
main :-
     cnt:decrease(1),
     cnt:current_value(X),
     display(X).
```

Notice that dynamically expanded goals such as `X=cnt,X:decrease(1)` will not work, use `X=counter(cnt),X:decrease(1)` instead.

Usage: :- `instance_of(Object,Constructor)`.

- *Description:* Declares `Object` to be an instance of the class denoted by `Constructor`.
- *The following properties should hold at call time:*

  `Object` is an unique term which identifies an object. (objects_rt:instance_id/1)

  `Constructor` is a term whose functor matches a class name. (objects_rt:constructor/1)

new/2: DECLARATION

This declaration has the same effect as `instance_of/2`.

Usage: :- `new(Object,Constructor)`.

- *Description:* Just an alias for `instance_of/2`.
- *The following properties should hold at call time:*

  `Object` is an unique term which identifies an object. (objects_rt:instance_id/1)

  `Constructor` is a term whose functor matches a class name. (objects_rt:constructor/1)

119.3 Other information (`objects_doc`)

Compile-time errors are restricted to some local analysis. Since there is no type declaration in the Prolog language, there is no posibility to determine whenever a given variable will hold an instance of any class.

However, little semantic analysis is performed. User may aid to perform such an analysis by the usage of run time checks (which are also detected at compile time), or static declarations. For example:

```
clause(Obj) :- Obj:a_method(334).
```

O'Ciao may be not able to determine whenever a_method/1 is a valid method for instance Obj, unless some help is provided:

```
clause(Obj) :- Obj instance_of myclass,Obj:a_method(334).
```

In such case, O'Ciao will report any semantic error at compile-time.

Most of the run-time errors are related to normal Ciao Prolog module system. Since objects are treated as normal Prolog modules at run time, there is no further documentation here about that stuff.

119.3.1 Error reporting at compile time (objects)

- **ERROR : invalid instance identifier** *ID***: must be an atom**
  There is a `instance_of/2` or `new/2` declaration where first argument *ID* must be an unique atom, but currently it is not. Statically declared instances needs an identifier to be provided by the user.
- **ERROR : instance identifier** *ID* **already in use**
  There are two or more `instance_of/2` declarations with the same first argument *ID*. Instance identifiers must be unique.
- **ERROR : invalid use_class/1 declaration:** *SourceFile* **is not a class**
  Those are the causes for this error:
  - The given *SourceFile* does not exist, or is not accesible.
  - The given *SourceFile* is not a Prolog source.
  - The given *SourceFile* is a valid Prolog source, but it does not declare a class.
- **ERROR : unknown class on** *ID* **instance declaration**
  The class defined on the `instance_of/2` declaration for *ID* instance has not been loaded by a `use_class/1` declaration.
- **ERROR : instance identifier** *ID* **is an exisisting Prolog module**
  There is an statically declared instance whose identifier may cause interference with the Ciao Prolog module system. Use another instance identifier.
- **ERROR : unknown constructor on** *ID* **instance declaration**
  The given constructor on the `instance_of/2` declaration for *ID* has not been defined at the corresponding class.
- **ERROR : constructor is needed on** *ID* **instance declaration**
  No constructor was defined on the `instance_of/2` declaration for *ID* and default constructor is not allowed. You must provide a constructor.
- **ERROR : static instance** *ID* **was derived from a different constructor at** *AnotherModule*
  ID has been declared to be an static instance both on *AnotherModule* and current source, but different constructors were used. The most probable causes for this error are:
  - Occasionally, there is another module using the same instance identifier and it was not noticed by you. Another different identifier may be used instead.
  - It was you intention to use the same object as declared by the other module. In this case, the same constructor must be used.
- **ERROR : invalid first argument in call to new(**_Arg_**,_)**
  There is a new/1 goal in your code where first argument is not a free variable. For example:
  `myobj new myclass`
  First argument must be a variable in order to receive a run-time generated object identifier.
- **ERROR : unknown class in call to new(?,**_Constructor_**)**
  The given *Constructor* in call to new/2 does not correspond to any used class at current code. The most probable cause of this may be:

  - You forgot to include a `use_class/1` declaration in your code.
  - There is a spelling mistake in the constructor.For example:
    :- use_class(myclass).
    foo(X) :- X new mclass.
- **ERROR : can not create an instance from an interface: new(?,*Constructor*)**
  Given *Constructor* references an interface rather than a class. Instances can not be derived from interface-expanded code.
- **ERROR : unknown constructor in call to new(?,*Constructor*)**
  As the previous error, there is a mistake in the given *Constructor*. This error is reported when you are trying to call a constructor which was not defined at the corresponding class. Check the class definition to find what is going on.
  Another cause for this error is the incorrect usage of the default constructor. Whenever there are one or more constructors defined at the involved class, you are restricted to chose one of them. This seems that default constructor will be available, if and only if, there are no constructors defined at the involved class.
- **ERROR : call to non-public** *ID:Goal*
  You are trying to call a method which was not declared as public by the class specified in `instance_of/2` declaration for *ID*.
- **ERROR : call to inaccessible predicate at instance** *ID:Goal*
  There is a call to *Goal* at statically declared instance *ID* which is unknown or was not declared as public.
- **ERROR : unknown instance** *ID* **of class** *Class* **at** *Goal*
  There is a call to *Goal* where involved statically declared instance *ID* is unknown or is not derived from *Class*. Check whether it was declared by a `instance_of/2` declaration.
- **ERROR : inaccessible attribute** *Fact* **at instance** *ID*
  There is an attempt to use *ID:Fact* but it was not declared as public.
- **ERROR : unknown attribute** *Fact* **at instance** *ID*
  There is an attempt to use *ID:Fact* but it is unknown or it is not an attribute (may be a method).
- **WARNING : invalid call to new(?,_)**
  There is a call to new/2 in you code where first argument variable has been determined to hold any other instance. For example:
  `foo :- X new myclass,X new otherclass.`
  or
  `foo(X) :- X instance_of myclass, X new myclass.`
  The related call to new/2 will allways fail.
- **WARNING : called** *Goal* **is not public at any used class**
  There is a call to *Var*:*Goal* where *Var* has not been determined to be compatible with any class. However, *Goal* is not public at any class specified by the `use_class/1` declaration.
  This is a warning (not an error) since *Var*:*Goal* may be not related to Object Oriented Programing.

119.3.2 Error reporting at run time (objects)

- **EXCEPTION : instantiation_error('1st argument must be free variable')**
  Calling to new/1 requieres first argument to be a free variable. For example:
  `X = this_will_raise_an_exception,X new myclass.`

- **EXCEPTION : instantiation_error('class not given')**
  You called new/2 using a free variable as second argument.
- **EXCEPTION : instantiation_error(inaccesible_class(***Class***), from(***Module***))**
  Module tried to create an instance of *Class* by the ways of new/2, but there is no usage relationship between *Module* and *Class*.
- **EXCEPTION : instantiation_error(invalid_constructor(** *Constructor***))**
  Constructor was not defined by the corresponding class.

120 Run time usage of objects

Author(s): Angel Fernandez Pineda, Angel Fernandez Pineda.

This library provides run-time support for object creation and manipulation. Objects are also called class instances, or simply instances.

Objects in Ciao are treated as normal modules. This is, an object is a run-time generated Prolog module, which may be identified by an unique term across the whole application.

This is a very simple example of how to create an instance, and how to make calls to it:

```
AnObj new myclass,
AnObj:mymethod.
```

In order to make any object accessible from code, an usage relationship must be established between the class (from which instances are derived) and the code itself. Refer to `use_class/1` predicate or `use_class/1` declaration in order to do so.

120.1 Usage and interface (`objects_rt`)

- **Library usage:**

  This library is automatically loaded when using the *objects* package:

  ```
  :- module(ModuleName,Exports,[objects]).
  ```

  Nothing special needs to be done.
- **Exports:**
  - *Predicates:*

    `new/2`, `instance_of/2`, `derived_from/2`, `interface/2`, `instance_codes/2`, `destroy/1`, `use_class/1`.
  - *Properties:*

    `constructor/1`, `class_name/1`, `interface_name/1`, `instance_id/1`, `class_source/1`, `interface_source/1`, `method_spec/1`, `virtual_method_spec/1`.
- **Other modules used:**
  - *System library modules:*

    `compiler/compiler`, `prolog_sys`.

120.2 Documentation on exports (`objects_rt`)

new/2: PREDICATE

Dynamic instance creation takes place by the ways of this predicate.

It takes a free variable as first argument which will be instantiated to an internal object identifier.

Second argument must be instantiated to a class constructor. Class constructors are designed to perform an initialization on the new created instance. Notice that instance initialization may involve some kind of computation, not only *state initialization*.

A class constructor is made by a functor, which must match the intended class name, and any number of parameters. For example:

```
Obj new myclass(1500,'hello, world!!!')
```

Those parameters depends (obviously) on the constructors defined at the class source. If no constructors where defined, no parameters are needed. This is called the default constructor. An example:

```
Obj new myclass
```

The default constructor can not be called if there is any constructor available at the class source.

Instantiation will raise an exception and fail whenever any of this conditions occur:

- First argument is not a free variable.
- Second argument functor is a class, but there is no usage relationship with it.
- Second argument functor is not a class.
- The given constructor is unknown.
- The given constructor fails (notice that default constructor never fails).

Objects may also be statically declared, refer to `instance_of/2` declaration.

Usage: `new(InstanceVar,Constructor)`

– *Description:* Creates a new instance of the class specified by `Constructor` returning its identifier in `InstanceVar`

– *The following properties should hold at call time:*

`InstanceVar` is a free variable. (term_typing:var/1)

`Constructor` is a term whose functor matches a class name. (objects_rt:constructor/1)

– *The following properties should hold upon exit:*

`InstanceVar` is an unique term which identifies an object. (objects_rt:instance_id/1)

instance_of/2: PREDICATE

This predicate is used to perform dynamic type checking. You may check whether a particular instance belongs to a particular class or related descendants.

instance_of/2 is used to perform static semantic analisys over object oriented code constructions.

By the use of instance_of/2 you may help to perform such analisys.

Usage 1: `instance_of(Instance,Class)`

– *Description:* Test whether `Instance` was derived from any descendant of `Class`, or that class itself

– *The following properties should hold at call time:*

`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)

`Class` is an atom denoting a class. (objects_rt:class_name/1)

Usage 2: `instance_of(Instance,Class)`

– *Description:* Retrieves, on backtracking, the inheritance line of `Instance` commencing on the creation class (that specified on call to `new/2`) and continuing on the rest of ascendant classes, if any.

– *The following properties should hold at call time:*

`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)

`Class` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*
`Class` is an atom denoting a class. (objects_rt:class_name/1)

derived_from/2: PREDICATE

Test whether an object identifier was derived directly from a class, by the usage of `new/2` or a static instance declaration (`instance_of/2`).

Usage 1: `derived_from(Instance,Class)`

– *Description:* Test derivation of `Instance` from `Class`
– *The following properties should hold at call time:*
`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)
`Class` is an atom denoting a class. (objects_rt:class_name/1)

Usage 2: `derived_from(Instance,Class)`

– *Description:* Retrieves the `Class` responsable of the derivation of `Instance`.
– *The following properties should hold at call time:*
`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)
`Class` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Class` is an atom denoting a class. (objects_rt:class_name/1)

interface/2: PREDICATE

This predicate is used to ensure a given interface to be implemented by a given instance.

Usage 1: `interface(Instance,Interface)`

– *Description:* Check whether `Instance` implements the given `Interface`.
– *The following properties should hold at call time:*
`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)
`Interface` is an unique atom which identifies a public interface. (objects_rt:interface_name/1)

Usage 2: `interface(Instance,Interfaces)`

– *Description:* Retrieves on backtracking all the implemented `Interfaces` of `Instance`.
– *The following properties should hold at call time:*
`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)
`Interfaces` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Interfaces` is an unique atom which identifies a public interface. (objects_rt:interface_name/1)

instance_codes/2: PREDICATE

Retrieves a character string representation from an object identifier and vice-versa.

Usage 1: `instance_codes(Instance,String)`

– *Description:* Retrieves a `String` representation of given `Instance`.

– *The following properties should hold at call time:*

`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)

`String` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`String` is a string (a list of character codes). (basic_props:string/1)

Usage 2: `instance_codes(Instance,String)`

– *Description:* Reproduces an `Instance` from its `String` representation. Such an instance must be alive across the application: this predicate will fail whether the involved instance has been destroyed.

– *The following properties should hold at call time:*

`Instance` is a free variable. (term_typing:var/1)

`String` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold upon exit:*

`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)

destroy/1: PREDICATE

As well as instances are created, they must be destroyed when no longer needed in order to release system resources.

Unfortunately, current O'Ciao implementation does not support automatic instance destruction, so user must manually call *destroy/1* in order to do so.

The programmer **must ensure** that no other references to the involved object are left in memory when destroy/1 is called. If not, unexpected results may be obtained.

Usage: `destroy(Instance)`

– *Description:* Destroys the object identified by `Instance`.

– *The following properties should hold at call time:*

`Instance` is an unique term which identifies an object. (objects_rt:instance_id/1)

use_class/1: PREDICATE

The behaviour of this predicate is identical to that provided by the declaration of the same name `use_class/1`. It allows user programs to dynamically load classes. Whether the given source is not a class it will perform a `use_module/1` predicate call.

Usage: `use_class(ClassSource)`

– *Description:* Dynamically loads the given `ClassSource`

– *The following properties should hold at call time:*

`ClassSource` is a valid path to a prolog file containing a class declaration (without .pl extension). (objects_rt:class_source/1)

constructor/1: PROPERTY

Usage: `constructor(Cons)`

– *Description:* `Cons` is a term whose functor matches a class name.

class_name/1: PROPERTY

Usage: `class_name(ClassName)`

– *Description:* `ClassName` is an atom denoting a class.

interface_name/1: PROPERTY

Usage: `interface_name(Interface)`

– *Description:* `Interface` is an unique atom which identifies a public interface.

instance_id/1: PROPERTY

Usage: `instance_id(ID)`

– *Description:* `ID` is an unique term which identifies an object.

class_source/1: PROPERTY

Usage: `class_source(Source)`

– *Description:* `Source` is a valid path to a prolog file containing a class declaration (without .pl extension).

interface_source/1: PROPERTY

Usage: `interface_source(Source)`

– *Description:* `Source` is a valid path to a prolog file containing a class declaration or an interface declaration (without .pl extension).

method_spec/1: PROPERTY

There is no difference between method or attribute specifications, and habitual predicate specifications. It is just a Functor/Arity term.

Usage: `method_spec(Spec)`

– *Description:* `Spec` is a method or attribute specification.

virtual_method_spec/1: PROPERTY

Usage: `virtual_method_spec(Spec)`

– *Description:* `Spec` is a method specification.

120.3 Known bugs and planned improvements (`objects_rt`)

- Usage of objects from the `user` module does not work properly. It is better to use the `objects` package in a (proper) module.
- Not really a bug: when loading code which declares static instances from the toplevel shell, predicate `use_module/1`) will not work properly: those instances may be not correctly created, and predicates will fail whenever they are not supposed to do. This may be avoided by reloading again the involved module, but make sure it is modified and saved to disk before doing so.

121 Declaring abstract interfaces for classes

Author(s): Angel Fernandez Pineda.

O'CIAO abstract interfaces are simple modules which declares exported predicates with no implementation. The implementation itself will be provided by some class using an `implements/1` declaration.

O'CIAO classes may be also treated as interfaces just ignoring all exported predicate implementation.

In order to get information about error reporting, consult the `class_doc` chapter on this documentation.

121.1 Usage and interface (`interface_doc`)

- **Library usage:**
  To declare an interface, the *interface source expansion* package must be loaded:

```
:- interface(ItfName).
```

  or using a `module/3` declaration, as follows:

```
:- module(ItfName,[],[interface]).
```

  Note: interfaces does not declare any code, so there is no need to load them from the CIAO toplevel shell.

PART VIII - Interfaces to other languages and systems

Author(s): The CLIP Group.

The following interfaces to/from Ciao are documented in this part:

- External interface (e.g., to C).
- Socket interface.
- Tcl/tk interface.
- Web interface (http, html, xml, etc.);
- Persistent predicate databases (interface between the Ciao internal database and the external file system).
- SQL-like database interface (interface between the Ciao internal database and external SQL/ODBC systems).
- Java interface.
- Calling emacs from Ciao.

122 Foreign Language Interface

Author(s): Jose F. Morales, Manuel Carro.

Ciao Prolog includes a high-level, flexible way to interface C and Prolog, based on the use of assertions to declare what are the expected types and modes of the arguments of a Prolog predicate, and which C files contain the corresponding code. To this end, the user provides:

- A set of C files, or a precompiled shared library,
- A Ciao Prolog module defining whith predicates are implemented in the C files and the types and modes of their arguments, and
- an (optional) set of flags required for the compilation of the files.

The Ciao Prolog compiler analyzes the Prolog code written by the user and gathers this information in order to generate automatically C "glue" code implementing the data translation between Prolog and C, and to compile the C code into dynamically loadable C object files, which are linked automatically when needed.

122.1 Declaration of Types

Each predicate implemented as a foreign C function must have accompanying declarations in the Ciao Prolog associated file stating the types and modes of the C function. A sample declaration for `prolog_predicate` which is implemented as `foreign_function_name` is:

```
:- true pred prolog_predicate(m1(Arg1), ... mN(ArgN)) ::
              type1 * ... * typeN +
              (foreign(foreign_function_name), returns(ArgR)).
```

where `m1`, ..., `mN` and `type1`, ..., `typeN` are respectively the modes and types of the arguments. `foreign_function_name` is the name of the C function implementing `prolog_predicate/N`, and the result of this function is unified with `ArgR`, which must be one of `Arg1` ... `ArgN`.

This notation can be simplified in several ways. If the name of the foreign function is the same as the name of the Ciao Prolog predicate, `foreign(foreign_function_name)` can be replaced by `foreign/0`. `returns(ArgR)` specifies that the result of the function corresponds to the `ArgR` argument of the Ciao Prolog predicate. If the foreign function does not return anything (or if its value is ignored), then `returns(ArgR)` must be removed. Note that `returns` cannot be used without `foreign`. A simplified, minimal form is thus:

```
:- true pred prolog_predicate(m1(Arg1), ... mN(ArgN)) ::
              type1 * ... * typeN + foreign.
```

122.2 Equivalence between Ciao Prolog and C types

The automatic translation between Ciao Prolog and C types is defined (at the moment) only for some simple but useful types. The translation to be performed is solely defined by the types of the arguments in the Ciao Prolog file (i.e., no inspection of the corresponding C file is done). The names (and meaning) of the types known for performing that translation are to be found in Chapter 123 [Foreign Language Interface Properties], page 661; they are also summarized below (Prolog types are on the left, and the corresponding C types on the right):

- num <-> double
- int <-> int
- atm <-> char *
- string <-> char * (with trailing zero)

- byte_list <-> char * (a buffer of bytes, with associated length)
- int_list <-> int * (a buffer of integers, with associated length)
- double_list <-> double * (a buffer of doubles, with associated length)
- address <-> void *

Strings, atoms, and lists of bytes are passed to (and from) C as dynamically (`ciao_malloc`) created arrays of characters (bytes). Those arrays are freed by Ciao Prolog upon return of the foreign function unless the property `do_not_free/2` is specified (see examples below). This caters for the case in which the C files save in a private state (either by themselves, or by a library function being called by them) the values passed on from Prolog. The type `byte_list/1` requires an additional property, `size_of/2`, to indicate which argument represents its size.

Empty lists of bytes and integers are converted into C `NULL` pointers, and vice versa. Empty strings (`[]`) and null atoms (') are converted into zero-length, zero-ended C strings (`""`). C `NULL` strings and empty buffers (i.e., buffers with zero length) are transformed into the empty list or the null atom (`''`).

Most of the work is performed by the predicates in the Chapter 125 [Foreign Language Interface Builder], page 669, which can be called explicitly by the user. Doing that is not usually needed, since the Ciao Prolog Compiler takes care of building glue code files an of compiling and linking whatever is necessary.

122.3 Equivalence between Ciao Prolog and C modes

The (prefix) `+/1` ISO mode (or, equivalently, the in/1 mode) states that the corresponding Prolog argument is ground at the time of the call, and therefore it is an input argument in the C part; this groundness is automatically checked upon entry. The (prefix) `-/1` ISO mode (or, equivalently, the go/1 mode) states that Prolog expects the C side to generate a (ground) value for that argument. Arguments with output mode should appear in C functions as pointers to the corresponding base type (as it is usual with C), i.e., an argument which is an integer generated by the C file, declared as

```
:- true pred get_int(go(ThisInt)) :: int + foreign
```

or as

```
:- true pred get_int(-ThisInt) :: int + foreign
```

should appear in the C code as

```
void get_int(int *thisint)
{
        ....
}
```

Note the type of the (single) argument of the function. Besides, the return value of a function can always be used as an output argument, just by specifying to which Prolog arguments it corresponds, using the `foreing/1` property. The examples below illustrate this point, and the use of several assertions to guide the compilation.

122.4 Custom access to Prolog from C

Automatic type conversions does not cover all the possible cases. When the automatic type conversion is not enough (or if the user, for any reason, does not want to go through the automatic conversion), it is possible to instruct Ciao Prolog not to make implicit type conversion. The strategy in that case is to pass the relevant argument(s) with a special type (a `ciao_term`) which can represent any term which can be built in Prolog. Operations to construct, traverse, and test this data abstraction from C are provided. The prototypes of these operations are placed on the `"ciao_prolog.h"` file, under the `include` subdirectory of the installation directory

(the Ciao Prolog compiler knowns where it has been installed, and gives the C compiler the appropriate flags). This *non direct correspondence* mode is activated whenever a Ciao Prolog type unknown to the foreign interface (i.e., none of these in Chapter 123 [Foreign Language Interface Properties], page 661) or the type `any_term` (which is explicitly recognised by the foreign language interface) is found. The latter is preferred, as it is much more informative, and external tools, as the the CiaoPP preprocessor, can take advantage of them.

122.4.1 Term construction

All term construction primitives return an argument of type `ciao_term`, which is the result of constructing a term. All Ciao Prolog terms can be built using the interface operations `ciao_var()`, `ciao_structure()`, `ciao_integer()`, and `ciao_float()`. There are, however, variants and specialized versions of these operations which can be freely intermixed. Using one version or another is a matter of taste and convenience. We list below the prototypes of the primitives in order of complexity.

Throughout this section, **true**, when referred to a boolean value, correspond to the integer value `1`, and **false** correspond to the integer value `0`, as is customary in C boolean expressions. These values also available as the (predefined) constants `ciao_true` and `ciao_false`, both of type `ciao_bool`.

- `ciao_term ciao_var();`

  Returns a fresh, unbound variable.
- `ciao_term ciao_integer(int i);`

  Creates a term, representing an integer from the Prolog point of view, from a C integer.
- `ciao_term ciao_float(double i);`

  Creates a term, representing a floating point number, from a floating point number.
- `ciao_term ciao_put_number_chars(char *number_string);`

  It converts `number_string` (which must a string representing a syntactically valid number) into a `ciao_term`.
- `ciao_term ciao_atom(char *name);`

  Creates an atom whose printable name is given as a C string.
- `ciao_term ciao_structure_a(char *name, int arity, ciao_term *args);`

  Creates a structure with name 'name' (i.e., the functor name), arity 'arity' and the components of the array 'args' as arguments: `args[0]` will be the first argument, `args[1]` the second, and so on. The 'args' array itself is not needed after the term is created, and can thus be a variable local to a procedure. An atom can be represented as a 0-arity structure (with `ciao_structure(name, 0)`), and a list cell can be constructed using the '.'/2 structure name. The `_a` suffix stands for *array*.
- `ciao_term ciao_structure(char *name, int arity, ...);`

  Similar to ciao_structure_a, but the C arguments after the arity are used to fill in the arguments of the structure.
- `ciao_term ciao_list(ciao_term head, ciao_term tail);`

  Creates a list from a `head` and a `tail`. It is equivalent to `ciao_structure(".", 2, head, tail)`.
- `ciao_term ciao_empty_list();`

  Creates an empty list. It is equivalent to `ciao_atom("[]")`.
- `ciao_term ciao_listn_a(int len, ciao_term *args);`

  Creates a list with 'len' elements from the array `args`. The *nth* element of the list (starting at 1) is `args[n-1]` (starting at zero).

- `ciao_term ciao_listn(int length, ...);`
  Like `ciao_listn_a()`, but the list elements appear explicitly as arguments in the call.
- `ciao_term ciao_dlist_a(int len, ciao_term *args, ciao_term base);`
  Like `ciao_listn_a`, but a difference list is created. `base` whill be used as the tail of the list, instead of the empty list.
- `ciao_term ciao_dlist(int length, ...);`
  Similar to `ciao_dlist_a()` with a variable number of arguments. The last one is the tail of the list.
- `ciao_term ciao_copy_term(ciao_term src_term);`
  Returns a new copy of the `term`, with fresh variables (as `copy_term/2` does).

122.4.2 Testing the Type of a Term

A `ciao_term` can contain *any* Prolog term, and its implementation is opaque to the C code. Therefore the only way to know reliably what data is passed on is using explicit functions to test term types. Below, `ciao_bool` is a type defined in `"ciao_prolog.h"` which can take the values 1 (for **true**) and 0 (for **false**).

- `ciao_bool ciao_is_variable(ciao_term term);`
  Returns true if `term` is currently an uninstantiated variable.
- `ciao_bool ciao_is_number(ciao_term term);`
  Returns true if `term` is an integer (of any length) or a floating point number.
- `ciao_bool ciao_is_integer(ciao_term term);`
  Returns true if `term` is instantiated to an integer.
- `ciao_bool ciao_fits_in_int(ciao_term term);`
  Returns true if `term` is instantiated to an integer which can be stored in an `int`, and false otherwise.
- `ciao_bool ciao_is_atom(ciao_term atom);`
  Returns true if `term` is an atom.
- `ciao_bool ciao_is_list(ciao_term term);`
  Returns true if `term` is a list (actually, a `cons` cell).
- `ciao_bool ciao_is_empty_list(ciao_term term);`
  Returns true if `term` is the atom which represents the empty list (i.e., `[]`).
- `ciao_bool ciao_is_structure(ciao_term term);`
  Returns true if `term` is a structure of any arity. This includes atoms (i.e., structures of arity zero) and lists, but excludes variables and numbers.

122.4.3 Term navigation

The functions below can be used to recover the value of a `ciao_term` into C variables, or to inspect Prolog structures.

- `int ciao_to_integer(ciao_term term);`
  Converts `term` to an integer. `ciao_is_integer(term)` must hold.
- `ciao_bool ciao_to_integer_check(ciao_term term, int *result);`
  Checks whether `term` fits into the size of an integer. If so, true is returned and `*result` is unified with the integer `term` represents. Otherwise, false is returned and `*result` is not touched.

- `double ciao_to_float(ciao_term term);`
  Converts `term` to a float value. `ciao_is_number(term)` must hold.
- `char *ciao_get_number_chars(ciao_term term);`
  It converts `ciao_term` (which must be instantiated to a number) into a C string representing the number in the current radix. The string returned is a copy, which must (eventually) be explicitly deallocated by the user C code using the operation `ciao_free()`
- `char *ciao_atom_name(ciao_term atom);`
  Returns the name of the atom. The returned string *is the one internally used by Ciao Prolog*, and should not be deallocated, changed or altered in any form. The advantage of using it is that it is fast, as no data copying is needed.
- `char *ciao_atom_name_dup(ciao_term atom);`
  Obtains a **copy** of the name of the atom. The string can be modified, and the programmer has the responsibility of deallocating it after being used. Due to the copy, it is slower than calling `char *ciao_atom_name()`.
- `ciao_term ciao_list_head(ciao_term term);`
  Extracts the head of the list `term`. Requires `term` to be a list.
- `ciao_term ciao_list_tail(ciao_term term);`
  Extracts the tail of the list `term`. Requires `term` to be a list.
- `char *ciao_structure_name(ciao_term term);`
  Extracts the name of the structure `term`. Requires `term` to be a structure.
- `int ciao_structure_arity(ciao_term term);`
  Extracts the arity of the structure `term`.
  Requires `term` to be a structure.
- `ciao_term ciao_structure_arg(ciao_term term, int n);`
  Extracts the *nth* argument of the structure `term`. It behaves like `arg/3`, so the first argument has index 1. Requires `term` to be a structure.

122.4.4 Testing for Equality and Performing Unification

Variables of type `ciao_term` cannot be tested directly for equality: they are (currently) implemented as a sort of pointers which may be aliased (two different pointers may refer to the same object). The interface provides helper functions for testing term equality and to perform unification of terms.

- `ciao_bool ciao_unify(ciao_term x, ciao_term y);`
  Performs the unification of the terms `x` and `y`, and returns true if the unification was successful. This is equivalent to calling the (infix) Prolog predicate `=/2`. The bindings are trailed and undone on backtracking.
- `ciao_bool ciao_equal(ciao_term x, ciao_term y);`
  Performs equality testing of terms, and returns true if the test was successful. This is equivalent to calling the (infix) Prolog predicate `==/2`. Equality testing does not modify the terms compared.

122.4.5 Raising Exceptions

The following functions offers a way of throwing exceptions from C that can be caught in Prolog with `catch/3`. The term that reaches Prolog is exactly the same which was thrown by C. The execution flow is broken at the point where `ciao_raise_exception()` is executed, and it returns to Prolog.

- `void ciao_raise_exception(ciao_term ball);`
  Raises an exception an throws the term `ball`.

122.4.6 Creating and disposing of memory chunks

Memory to be used solely by the user C code can be reserved/disposed of using, e.g., the well-known `malloc()`/`free()` functions (or whatever other functions the user may have available). However, memory explicitly allocated by Ciao Prolog and passed to C code, or allocated by C code and passed on to Ciao Prolog (and subject to garbage collection by it) should be allotted and freed (when necessary) by using the functions:

- `void *ciao_malloc(int size);`
- `void ciao_free(void *pointer);`

whose behavior is similar to `malloc()`/`free()`, but which will cooordinate properly with Ciao Prolog's internal memory management.

122.4.7 Calling Prolog from C

It is also possible to make arbitraty calls to Prolog predicates from C. There are two basic ways of make a query, depending on whether only one solution is needed (or if the predicate to be called is known to generate only one solution), or if several solutions are required.

When only one solution is needed `ciao_commit_call` obtains it (the solution obtained will obviously be the first one) and discards the resources used for finding it:

- `ciao_bool ciao_commit_call(char *name, int arity, ...);`

  Makes a call to a predicate and returns true or false depending on whether the query has succedeed or not. In case of success, the (possibly) instantiated variables are reachable from C.
- `ciao_bool ciao_commit_call_term(ciao_term goal);`

  Like `ciao_commit_call()` but uses the previously built term `goal` as goal.

If more than one solution is needed, it is necessary to use the `ciao_query` operations. A consult begins with a `ciao_query_begin` which returns a `ciao_query` object. Whenever an additional solution is required, the `ciao_query_next` function can be called. The query ends by calling `ciao_query_end` and all pending search branches are pruned.

- `ciao_query *ciao_query_begin(char *name, int arity, ...);`

  The predicate with the given name, arity and arguments (similar to the `ciao_structure()` operation) is transformed into a `ciao_query` object which can be used to make the actual query.
- `ciao_query *ciao_query_begin_term(ciao_term goal);`

  Like ciao_query_begin but using the term `goal` instead.
- `ciao_bool ciao_query_ok(ciao_query *query);`

  Determines whether the query may have pending solutions. A false return value means that there are no more solutions; a true return value means that there are more possible solutions.
- `void ciao_query_next(ciao_query *query);`

  Ask for a new solution.
- `void ciao_query_end(ciao_query *query);`

  Ends the query and frees the used resources.

122.5 Examples

122.5.1 Mathematical functions

In this example, the standard mathematical library is accessed to provide the *sin*, *cos*, and *fabs* functions. Note that the library is specified simply as

```
:- use_foreign_library([m]).
```

The foreign interface adds the `-lm` at compile time. Note also how some additional options are added to optimize the compiled code (only glue code, in this case) and mathematics (only in the case of Linux in an Intel processor).

File *math.pl*:

```
:- module(math, [sin/2, cos/2, fabs/2], [foreign_interface]).

:- true pred sin(in(X),go(Y)) :: num * num + (foreign,returns(Y)).
:- true pred cos(in(X),go(Y)) :: num * num + (foreign,returns(Y)).
:- true pred fabs(in(X),go(Y)) :: num * num + (foreign,returns(Y)).

:- extra_compiler_opts(['-O2']).
:- extra_compiler_opts('LINUXi86',['-ffast-math']).
:- use_foreign_library('LINUXi86', m).
```

122.5.2 Addresses and C pointers

The `address` type designates any pointer, and provides a means to deal with C pointers in Prolog without interpreting them whatsoever. The C source file which implements the operations accessed from Prolog is declared with the

```
:- use_foreign_source(objects_c).
```

directive.

File *objects.pl*:

```
:- module(objects, [object/2, show_object/1], [foreign_interface]).

:- true pred object(in(N),go(Object)) ::
        int * address + (foreign,returns(Object)).

:- true pred show_object(in(Object)) ::
        address + foreign.

:- use_foreign_source(objects_c).
:- extra_compiler_opts('-O2').
```

File *objects_c.c*:

```
#include <stdio.h>

struct object {
  char *name;
  char *colour;
};

#define OBJECTS 3

struct object objects[OBJECTS] =
```

```
{ {"ring","golden"},
  {"table","brown"},
  {"bottle","green"} };

struct object *object(int n) {
  return &objects[n % OBJECTS];
}

void show_object(struct object *o) {
  printf("I show you a %s %s\n", o->colour, o->name);
}
```

122.5.3 Lists of bytes and buffers

A list of bytes (c.f., a list of ints) corresponds to a byte buffer in C. The length of the buffer is associated to that of the list using the property `size_of/2`. The returned buffer **is freed by Ciao Prolog** upon its recepction, unless the `do_not_free/1` property is specified (see later). Conversely, a list of natural numbers in the range 0 to 255 can be passed to C as a buffer.

File *byte_lists.pl*:

```
:- module(byte_lists, [obtain_list/3, show_list/2], [foreign_interface]).

:- true pred obtain_list(in(N),go(Length),go(List)) :: int * int * byte_list
        + (foreign,size_of(List,Length)).
:- true pred show_list(in(Length),in(List)) :: int * byte_list
        + (foreign,size_of(List,Length)).

:- use_foreign_source(bytes_op).
```

File *bytes_op.c*:

```
#include <stdlib.h>
#include <stdio.h>

void obtain_list(int n, int *l, char **s) {
  int i;
  int c;
  if (n < 0) n = 0;
  *l = n;
  *s = (char *)malloc(*l);
  for (i = 0; i < *l; i++) {
    (*s)[i] = i;
  }
}

void show_list(int l, char *s) {
  if (s) {
    int n;
    printf("From C:");
    for (n = 0; n < l; n++) {
      printf(" %d", s[n]);
```

```
    }
    printf(".\n");
  } else {
    printf("From C: []\n");
  }
}
```

122.5.4 Lists of integers

File *int_lists.pl*:

```
:- module(int_lists, [obtain_list/3, show_list/2], [foreign_interface]).

:- true pred obtain_list(in(N),go(Length),go(List)) :: int * int * int_list
        + (foreign,size_of(List,Length)).
:- true pred show_list(in(Length),in(List)) :: int * int_list
        + (foreign,size_of(List,Length)).

:- use_foreign_source(ints_op).
```

File *ints_op.c*:

```
#include <stdlib.h>
#include <stdio.h>

void obtain_list(int n, int *l, int **s) {
  int i;
  int c;
  if (n < 0) n = 0;
  *l = n;
  *s = (int *)malloc((*l) * sizeof(int));
  for (i = 0; i < *l; i++) {
    (*s)[i] = i;
  }
}

void show_list(int l, int *s) {
  if (s) {
    int n;
    printf("From C:");
    for (n = 0; n < l; n++) {
      printf(" %d", s[n]);
    }
    printf(".\n");
  } else {
```

```
    printf("From C: []\n");
  }
}
```

122.5.5 Strings and atoms

A C string can be seen as a buffer whose end is denoted by the trailing zero, and therefore stating its length is not needed. Two translations are possible into Ciao Prolog: as a Prolog string (list of bytes, with no trailing zero) and as an atom. These are selected automatically just by choosing the corresponding type (look at the examples below).

Note how the `do_not_free/1` property is specified in the `a_string/1` predicate: the string returned by C is static, and therefore it should not be freed by Prolog.

File *strings_and_atoms.pl*:

```
:- module(strings_and_atoms,
        [ lookup_string/2,
          lookup_atom/2,
          a_string/1,
          show_string/1,
          show_atom/1
        ],
        [foreign_interface]).

:- true pred a_string(go(S)) ::
        string + (foreign(get_static_str),returns(S),do_not_free(S)).

:- true pred lookup_string(in(N),go(S)) ::
        int * string + (foreign(get_str),returns(S)).
:- true pred lookup_atom(in(N),go(S)) ::
        int * atm + (foreign(get_str),returns(S)).

:- true pred show_string(in(S)) :: string + foreign(put_str).
:- true pred show_atom(in(S)) :: atm + foreign(put_str).

:- use_foreign_source(str_op).
```

File *str_op.c*:

```
#include <stdlib.h>
#include <stdio.h>

char *get_static_str() {
  return "this is a string Prolog should not free";
}
```

```
char *get_str(int n) {
  char *s;
  int size;
  int i;
  int c;
  if (n < 0) n = -n;
  size = (n%4) + 5;
  s = (char *)malloc(size+1);
  for (i = 0, c = ((i + n) % ('z' - 'a' + 1)) + 'a'; i < size; i++,c++) {
    if (c > 'z') c = 'a';
    s[i] = c;
  }
  s[i] = 0;
  return s;
}

void put_str(char *s) {
  if (s) {
    printf("From C: \"%s\"\n", s);
  } else {
    printf("From C: null\n");
  }
}
```

122.5.6 Arbitrary Terms

This example shows how data Prolog can be passed untouched to C code, and how it can be manipulated there.

File *any_term.pl*:

```
:- module(any_term,
        [custom_display_term/1,
         custom_create_term/2
        ],
        [foreign_interface]).

:- true pred custom_display_term(in(X)) :: any_term + foreign.
:- true pred custom_create_term(in(L), go(X)) :: int * any_term + (foreign,returns(X

:- use_foreign_source(any_term_c).
:- extra_compiler_opts('-O2').
```

File *any_term_c.c*:

```
#include <stdio.h>
```

```
#include "ciao_prolog.h"

ciao_term custom_create_term(int n) {
  ciao_term t;
  t = ciao_empty_list();
  while (n > 0) {
    t = ciao_list(ciao_integer(n), t);
    n--;
  }
  return t;
}

void custom_display_term(ciao_term term) {
  if (ciao_is_atom(term)) {
    printf("<atom name=\"%s\"/>", ciao_atom_name(term));
  } else if (ciao_is_structure(term)) {
    int i;
    int a;
    a = ciao_structure_arity(term);
    printf("<structure name=\"%s\" arity=\"%d\">", ciao_structure_name(term), a);
    for (i = 1; i <= a; i++) {
      printf("<argument number=\"%d\">", i);
      custom_display_term(ciao_structure_arg(term, i));
      printf("</argument>");
    }
    printf("</structure>");
  } else if (ciao_is_list(term)) {
    printf("<list>");
    printf("<head>");
    custom_display_term(ciao_list_head(term));
    printf("</head>");
    printf("<tail>");
    custom_display_term(ciao_list_tail(term));
    printf("</tail>");
    printf("</list>");
  } else if (ciao_is_empty_list(term)) {
    printf("<empty_list/>");
  } else if (ciao_is_integer(term)) {
    printf("<integer value=\"%d\"/>", ciao_to_integer(term));
  } else if (ciao_is_number(term)) {
    printf("<float value=\"%f\"/>", ciao_to_float(term));
  } else {
    printf("<unknown/>");
  }
}
```

122.5.7 Exceptions

The following example defines a predicate in C that converts a list of codes into a number using `strtol()`. If this conversion fails, then a exception is raised.

File *exceptions_example.pl*:

```
:- module(exceptions_example,
        [codes_to_number_c/2,
         safe_codes_to_number/2
        ],
        [foreign_interface]).

:- use_module(library(format)).

% If the string is not a number raises an exception.
:- true pred codes_to_number_c(in(X), go(Y)) :: string * int + (foreign, returns(Y))

safe_codes_to_number(X, Y) :-
        catch(codes_to_number_c(X, Y), Error, handle_exception(Error)).

handle_exception(Error) :- format("Exception caught ~w~n", [Error]).

:- use_foreign_source(exceptions_c).
:- extra_compiler_opts('-O2').
```

File *exceptions_c.c*:

```
#include <string.h>
#include "ciao_prolog.h"

int codes_to_number_c(char *s) {
  char *endptr;
  int n;
  n = strtol(s, &endptr, 10);
  if (endptr == NULL || *endptr != '\0') {
    ciao_raise_exception(ciao_structure("codes_to_number_exception",
                                        1,
                                        ciao_atom(s)));
  }
  return n;
}
```

122.5.8 Testing number types and using unbound length integers

Unbound length integers (and, in general, any number) can be converted to/from `ciao_terms` by using strings. The following examples show two possibilities: one which tries to be as smart as possible (checking whether numbers fit into a machine int or not), and being lazy and simpler -and probably slower.

File *bigints.pl*:

```
:- module(bigints,
```

```
        [
          make_smart_conversion/3, % Checks and uses convenient format
          force_string_conversion/2  % Passes around using strings
        ],
        [foreign_interface]).

:- true pred make_smart_conversion_c(in(X), go(Y), go(How))::
        any_term * any_term * any_term + foreign #
"Given a number @var{X}, it is unified with @var{Y} by using the most
specific internal representation (short integer, float, or long
integer).  @var{How} returns how the conversion was done.
It behaves unpredictably if @var{X} is not a number.".

:- true pred force_string_conversion_c(in(X), go(Y))::
        any_term * any_term + foreign #
"Given a number @var{X}, it is unified with @var{Y} by using the most
general internal representation (a string of characters). It behaves
unpredictably if @var{X} is not a number.".

:- use_foreign_source(bigints_c).

make_smart_conversion(A, B, C):-
        number(A),                              % Safety test
        make_smart_conversion_c(A, B, C).

force_string_conversion(A, B):-
        number(A),                              % Safety test
        force_string_conversion_c(A, B).
```

File *bigints_c.c*:

```
#include "ciao_prolog.h"

void make_smart_conversion_c(ciao_term  number_in,
                             ciao_term *number_out,
                             ciao_term *how_converted) {
  int    inter_int;
  double inter_float;
  char * inter_str;

  if (ciao_fits_in_int(number_in)) {/* Includes the case of being a float */
    inter_int = ciao_to_integer(number_in);
    *number_out = ciao_integer(inter_int);
    *how_converted = ciao_atom("machine_integer");
  } else
    if (ciao_is_integer(number_in)) { /* Big number */
      inter_str   = ciao_get_number_chars(number_in);
      *number_out = ciao_put_number_chars(inter_str);
      ciao_free(inter_str);
      *how_converted = ciao_atom("string");
    } else { /* Must be a float */
      inter_float = ciao_to_float(number_in);
```

```
      *number_out = ciao_float(inter_float);
      *how_converted = ciao_atom("float");
    }
}

void force_string_conversion_c(ciao_term  number_in,
                               ciao_term *number_out) {
  char *inter_str;
  inter_str  = ciao_get_number_chars(number_in);
  *number_out = ciao_put_number_chars(inter_str);
  ciao_free(inter_str);
}
```

122.6 Usage and interface (foreign_interface_doc)

- **Library usage:**
  The foreign interface is used by including `foreign_interface` in the include list of a module, or by means of an explicit `:- use_package(foreign_interface)`.

123 Foreign Language Interface Properties

Author(s): Jose F. Morales, Manuel Carro.

The foreign language interface uses some properties to specify linking regimes, foreign files to be compiled, types of data available, memory allocation policies, etc.

123.1 Usage and interface (`foreign_interface_properties`)

- **Library usage:**
  `:- use_module(library(foreign_interface_properties)).`
- **Exports:**
  - *Properties:*
    `foreign_low/1`, `foreign_low/2`, `size_of/3`, `foreign/1`, `foreign/2`, `returns/2`, `do_not_free/2`, `needs_state/1`, `ttr/3`.
  - *Regular Types:*
    `int_list/1`, `double_list/1`, `byte_list/1`, `byte/1`, `null/1`, `address/1`, `any_term/1`.

123.2 Documentation on exports (`foreign_interface_properties`)

int_list/1: REGTYPE

Usage: `int_list(List)`

– *Description:* `List` is a list of integers.

double_list/1: REGTYPE

Usage: `double_list(List)`

– *Description:* `List` is a list of numbers.

byte_list/1: REGTYPE

Usage: `byte_list(List)`

– *Description:* `List` is a list of bytes.

byte/1: REGTYPE

Usage: `byte(B)`

– *Description:* `Byte` is a byte.

null/1: REGTYPE

Usage: `null(Address)`

– *Description:* `Address` is a null adress.

address/1: REGTYPE

Usage: `address(Address)`

– *Description:* `Address` is a memory address.

any_term/1: REGTYPE

Usage: `any_term(X)`

– *Description:* `X` is any term. The foreign interface passes it to C functions as a general term.

foreign_low/1: PROPERTY

Usage: `foreign_low(Name)`

– *Description:* The Prolog predicate `Name` is implemented using the function `Name`. The implementation is not a common C one, but it accesses directly the internal Ciao Prolog data structures and functions, and therefore no glue code is generated for it.

foreign_low/2: PROPERTY

Usage: `foreign_low(PrologName,ForeignName)`

– *Description:* The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied.

size_of/3: PROPERTY

Usage: `size_of(Name,ListVar,SizeVar)`

– *Description:* For predicate `Name`, the size of the argument of type `byte_list/1`, `ListVar`, is given by the argument of type integer `SizeVar`.

foreign/1: PROPERTY

Usage: `foreign(Name)`

– *Description:* The Prolog predicate `Name` is implemented using the foreign function `Name`.

foreign/2: PROPERTY

Usage: `foreign(PrologName,ForeignName)`

– *Description:* The Prolog predicate `PrologName` is implemented using the foreign function `ForeignName`.

returns/2: PROPERTY

Usage: `returns(Name,Var)`

– *Description:* The result of the foreign function that implements the Prolog predicate `Name` is unified with the Prolog variable `Var`. Cannot be used without `foreign/1` or `foreign/2`.

do_not_free/2: PROPERTY

Usage: `do_not_free(Name,Var)`

– *Description:* For predicate `Name`, the C argument passed to (returned from) the foreign function will not be freed after calling the foreign function.

needs_state/1: PROPERTY

Usage: `needs_state(Name)`

– *Description:* The foreign function which implementes the predicate `Name` needs a `ciao_state` as its first argument.

ttr/3: PROPERTY

Usage: `ttr(Name,Var,TTr)`

– *Description:* For predicate `Name`, the C argument will be translated ussing `TTr` as term translator.

123.3 Documentation on internals (`foreign_interface_properties`)

use_foreign_source/1: DECLARATION

Usage: `:- use_foreign_source(Files).`

– *Description:* `Files` is the (list of) foreign file(s) that will be linked with the glue-code file. If the file(s) do(es) not have extension, then the '.c' extension will be automatically added

– *The following properties hold at call time:*

`Files` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

use_foreign_source/2: DECLARATION

Usage: `:- use_foreign_source(OsArch,Files).`

– *Description:* `Files` are the OS and architecture dependant foreign files. This allows compiling and linking different files depending on the O.S. and architecture.

– *The following properties hold at call time:*

`OsArch` is an atom. (basic_props:atm/1)

`Files` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

use_foreign_library/1: DECLARATION

Usage: `:- use_foreign_library(Libs).`

– *Description:* `Libs` is the (list of) external library(es) needed to link the C files. Only the short name of the library (i.e., what would follow the `-l` in the linker is needed.

– *The following properties hold at call time:*

`Libs` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

use_foreign_library/2: DECLARATION

Usage: :- `use_foreign_library(OsArch,Libs)`.

- *Description:* `Libs` are the OS and architecture dependant libraries.
- *The following properties hold at call time:*

  `OsArch` is an atom. (basic_props:atm/1)

  `Libs` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

extra_compiler_opts/1: DECLARATION

Usage: :- `extra_compiler_opts(Opts)`.

- *Description:* `Opts` is the list of additional compiler options (e.g., optimization options) that will be used during the compilation.
- *The following properties hold at call time:*

  `Opts` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

extra_compiler_opts/2: DECLARATION

Usage: :- `extra_compiler_opts(OsArch,Opts)`.

- *Description:* `Opts` are the OS and architecture dependant additional compiler options.
- *The following properties hold at call time:*

  `OsArch` is an atom. (basic_props:atm/1)

  `Opts` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

use_compiler/1: DECLARATION

Usage: :- `use_compiler(Compiler)`.

- *Description:* `Compiler` is the compiler to use in this file. When this option is used, the default (Ciao-provided) compiler options are not used; those specified in `extra_compiler_options` are used instead.
- *The following properties hold at call time:*

  `Compiler` is an atom. (basic_props:atm/1)

use_compiler/2: DECLARATION

Usage: :- `use_compiler(OsArch,Compiler)`.

- *Description:* `Compiler` is the compiler to use in this file when compiling for the architecture `OsArch`. The option management is the same as in `use_compiler/2`.
- *The following properties hold at call time:*

  `OsArch` is an atom. (basic_props:atm/1)

  `Compiler` is an atom. (basic_props:atm/1)

extra_linker_opts/1: DECLARATION

Usage: :- `extra_linker_opts(Opts)`.

- *Description:* `Opts` is the list of additional linker options that will be used during the linkage.
- *The following properties hold at call time:*

  `Opts` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

extra_linker_opts/2: DECLARATION

Usage: :- `extra_linker_opts(OsArch,Opts)`.

- *Description:* `Opts` are the OS and architecture dependant additional linker options.
- *The following properties hold at call time:*

  `OsArch` is an atom. (basic_props:atm/1)

  `Opts` is an atom or a list of atoms. (basic_props:atm_or_atm_list/1)

use_linker/1: DECLARATION

Usage: :- `use_linker(Linker)`.

- *Description:* `Linker` is the linker to use in this file. When this option is used, the default (Ciao-provided) linker options are not used; those specified in `extra_linker_options/1` are used instead.
- *The following properties hold at call time:*

  `Linker` is an atom. (basic_props:atm/1)

use_linker/2: DECLARATION

Usage: :- `use_linker(OsArch,Linker)`.

- *Description:* `Compiler` is the linker to use in this file when compiling for the architecture `OsArch`. The option management is the same as in `use_compiler/2`.
- *The following properties hold at call time:*

  `OsArch` is an atom. (basic_props:atm/1)

  `Linker` is an atom. (basic_props:atm/1)

foreign_inline/2: DECLARATION

Usage: :- `foreign_inline(Term,Text)`.

- *Description:* `Term` is a predicate name. `Text` is a source C code that define the predicate `Term`. `Term` is present for future use with the analyzers. Example of this can be viewed in the hrtimer library.
- *The following properties hold at call time:*

  `Term` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

  (basic_props:predname/1)

  `Text` is a string (a list of character codes). (basic_props:string/1)

123.4 Known bugs and planned improvements (`foreign_interface_properties`)

- The `size_of/3` property has an empty definition
- The `byte/1` property has an empty definition. A possible right definition is commented.

124 Utilities for on-demand compilation of foreign files

Author(s): Manuel Carro, Jose F. Morales.

This module provides two predicates which give the user information regarding how to compile external (C) files in order to link them with the Ciao Prolog engine at runtime.

These predicates are not intended to be called directly by the end-user. Instead, a tool or module whose aim is generating dynamically loadable files from source files should use the predicates in this file in order to find out what are the proper compiler and linker to use, and which options must be passed to them in the current architecture.

124.1 Usage and interface (foreign_compilation)

- **Library usage:**
  `:- use_module(library(foreign_compilation)).`
- **Exports:**
  - *Predicates:*
    `compiler_and_opts/2`, `linker_and_opts/2`.

124.2 Documentation on exports (foreign_compilation)

compiler_and_opts/2: PREDICATE

Usage: `compiler_and_opts(Compiler,Opts)`

- *Description:* If you want to compile a foreign language file for dynamic linking in the current operating system and architecture, you have to use the compiler `Compiler` and give it the options `Opts`. A variable in `Opts` means that no special option is needed.
- *Call and exit should be compatible with:*

  `Compiler` is an atom. (basic_props:atm/1)

  `Opts` is a list of `atm`s. (basic_props:list/2)

linker_and_opts/2: PREDICATE

Usage: `linker_and_opts(Linker,Options)`

- *Description:* If you want to link a foreign language file for dynamic linking in the current operating system and architecture, you have to use the linker `Compiler` and gite it the options `Opts`. A variable in `Opts` means that no special option is needed.
- *Call and exit should be compatible with:*

  `Linker` is an atom. (basic_props:atm/1)

  `Options` is a list of `atm`s. (basic_props:list/2)

125 Foreign Language Interface Builder

Author(s): Jose F. Morales, Manuel Carro.

Low-level utilities for building foreign interfaces. End-users should not need to use them, as the Ciao Prolog Compiler reads the user assertions and calls appropriately the predicates in this module.

125.1 Usage and interface (`build_foreign_interface`)

- **Library usage:**
  `:- use_module(library(build_foreign_interface)).`
- **Exports:**
  - *Predicates:*
    `build_foreign_interface/1`, `rebuild_foreign_interface/1`, `build_foreign_interface_explicit_decls/2`, `rebuild_foreign_interface_explicit_decls/2`, `build_foreign_interface_object/1`, `rebuild_foreign_interface_object/1`, `do_interface/1`.
- **Other modules used:**
  - *System library modules:*
    `write_c/write_c`, `streams`, `terms`, `lists`, `llists`, `aggregates`, `system`, `messages`, `assertions/assrt_lib`, `foreign_compilation`, `compiler/c_itf`, `compiler/engine_path`, `ctrlcclean`, `errhandle`, `filenames`.

125.2 Documentation on exports (`build_foreign_interface`)

build_foreign_interface/1: PREDICATE

Usage: `build_foreign_interface(File)`

- *Description:* Reads assertions from `File`, generates the gluecode for the Ciao Prolog interface, compiles the foreign files and the gluecode file, and links everything in a shared object. Checks modification times to determine automatically which files must be generated/compiled/linked.
- *Call and exit should be compatible with:*
  `File` is a source name. (streams_basic:sourcename/1)
- *The following properties should hold at call time:*
  `File` is currently ground (it contains no variables). (term_typing:ground/1)
- *The following properties should hold upon exit:*
  `File` is currently ground (it contains no variables). (term_typing:ground/1)

rebuild_foreign_interface/1: PREDICATE

Usage: `rebuild_foreign_interface(File)`

- *Description:* Like `build_foreign_interface/1`, but it does not check the modification time of any file.
- *Call and exit should be compatible with:*
  `File` is a source name. (streams_basic:sourcename/1)

– *The following properties should hold at call time:*
`File` is currently ground (it contains no variables). (term_typing:ground/1)
– *The following properties should hold upon exit:*
`File` is currently ground (it contains no variables). (term_typing:ground/1)

build_foreign_interface_explicit_decls/2: PREDICATE

Usage: `build_foreign_interface_explicit_decls(File,Decls)`

– *Description:* Like `build_foreign_interface/1`, but use declarations in `Decls` instead of reading the declarations from `File`.
– *Call and exit should be compatible with:*
`File` is a source name. (streams_basic:sourcename/1)
`Decls` is a list of `terms`. (basic_props:list/2)
– *The following properties should hold at call time:*
`File` is currently ground (it contains no variables). (term_typing:ground/1)
`Decls` is currently ground (it contains no variables). (term_typing:ground/1)
– *The following properties should hold upon exit:*
`File` is currently ground (it contains no variables). (term_typing:ground/1)
`Decls` is currently ground (it contains no variables). (term_typing:ground/1)

rebuild_foreign_interface_explicit_decls/2: PREDICATE

Usage: `rebuild_foreign_interface_explicit_decls(File,Decls)`

– *Description:* Like `build_foreign_interface_explicit_decls/1`, but it does not check the modification time of any file.
– *Call and exit should be compatible with:*
`File` is a source name. (streams_basic:sourcename/1)
`Decls` is a list of `terms`. (basic_props:list/2)
– *The following properties should hold at call time:*
`File` is currently ground (it contains no variables). (term_typing:ground/1)
`Decls` is currently ground (it contains no variables). (term_typing:ground/1)
– *The following properties should hold upon exit:*
`File` is currently ground (it contains no variables). (term_typing:ground/1)
`Decls` is currently ground (it contains no variables). (term_typing:ground/1)

build_foreign_interface_object/1: PREDICATE

Usage: `build_foreign_interface_object(File)`

– *Description:* Compiles the gluecode file with the foreign source files producing an unique object file.
– *Call and exit should be compatible with:*
`File` is a source name. (streams_basic:sourcename/1)
– *The following properties should hold at call time:*
`File` is currently ground (it contains no variables). (term_typing:ground/1)
– *The following properties should hold upon exit:*
`File` is currently ground (it contains no variables). (term_typing:ground/1)

rebuild_foreign_interface_object/1: PREDICATE

Usage: `rebuild_foreign_interface_object(File)`

- *Description:* Compiles (again) the gluecode file with the foreign source files producing an unique object file.
- *Call and exit should be compatible with:*

  `File` is a source name. (streams_basic:sourcename/1)
- *The following properties should hold at call time:*

  `File` is currently ground (it contains no variables). (term_typing:ground/1)
- *The following properties should hold upon exit:*

  `File` is currently ground (it contains no variables). (term_typing:ground/1)

do_interface/1: PREDICATE

Usage: `do_interface(Decls)`

- *Description:* Given the declarations in `Decls`, this predicate succeeds if these declarations involve the creation of the foreign interface
- *Call and exit should be compatible with:*

  `Decls` is a list of `terms`. (basic_props:list/2)
- *The following properties should hold at call time:*

  `Decls` is currently ground (it contains no variables). (term_typing:ground/1)
- *The following properties should hold upon exit:*

  `Decls` is currently ground (it contains no variables). (term_typing:ground/1)

126 Interactive Menus

Author(s): The CLIP Group.
This library package allows definition interactive menus

126.1 Usage and interface (`menu_doc`)

- **Library usage:**
  `:- use_package(menu).`
  or
  `:- module(...,...,[menu]).`
- **New operators defined:**
  `::/2` [970,xfx], `<-/2` [971,xfx], `guard/1` [900,fy], `$/2` [150,xfx], `=>/2` [950,xfx], `argnames/1` [1150,fx].
- **Other modules used:**
  - *System library modules:*
    `menu/menu_generator`, `menu/menu_rt`.

126.2 Documentation on multifiles (`menu_doc`)

menu_default/3: PREDICATE
No further documentation available for this predicate.
The predicate is *multifile*.

menu_opt/6: PREDICATE
No further documentation available for this predicate.
The predicate is *multifile*.

hook_menu_flag_values/3: PREDICATE
No further documentation available for this predicate.
The predicate is *multifile*.

hook_menu_check_flag_value/3: PREDICATE
No further documentation available for this predicate.
The predicate is *multifile*.

hook_menu_flag_help/3: PREDICATE
No further documentation available for this predicate.
The predicate is *multifile*.

hook_menu_default_option/3: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

127 menu_generator (library)

127.1 Usage and interface (menu_generator)

- **Library usage:**
  `:- use_module(library(menu_generator)).`
- **Exports:**
  - *Predicates:*
    `menu/1`, `menu/2`, `menu/3`, `menu/4`, `get_menu_flag/3`, `set_menu_flag/3`, `space/1`, `get_menu_configs/1`, `save_menu_config/1`, `remove_menu_config/1`, `restore_menu_config/1`, `show_menu_configs/0`, `show_menu_config/1`, `get_menu_options/2`, `get_menu_flags/1`, `restore_menu_flags_list/1`, `get_menu_flags/2`, `restore_menu_flags/2`, `generate_js_menu/1`, `eq/3`, `neq/3`, `uni_type/2`, `vmember/2`.
  - *Regular Types:*
    `menu_flag_values/1`.
  - *Multifiles:*
    `$is_persistent/2`, `persistent_dir/2`, `persistent_dir/4`, `menu_default/3`, `menu_opt/6`, `hook_menu_flag_values/3`, `hook_menu_check_flag_value/3`, `hook_menu_flag_help/3`, `hook_menu_default_option/3`.
- **Other modules used:**
  - *System library modules:*
    `persdb/persdbrt`, `aggregates`, `write`, `messages`, `prompt`, `lists`.

127.2 Documentation on exports (menu_generator)

menu/1: PREDICATE

Usage: `menu(M)`

- *Description:* Like menu(M , true).

menu/2: PREDICATE

Usage: `menu(M,Bool)`

- *Description:* Like `menu/4` with no selected options, taking the menu level from the term `M` (example: ana(1) is expert, ana is naive), and using `Bool` to decide whether print help message or not.

menu/3: PREDICATE

Usage: `menu(M,Level,Bool)`

- *Description:* Like `menu/4` with no selected options.

menu/4: PREDICATE

Usage: `menu(M,Level,Bool,AlreadySelectedOpts)`

– *Description:* Execute the menu `X`. `Level` specifies the menu level. `Bool` decides whether print the help message. `AlreadySelectedOpts` is a list with the selected options.

get_menu_flag/3: PREDICATE

Usage: `get_menu_flag(M,F,V)`

– *Description:* Returns the value in `V` of the flag `F` in the menu (-branch) `M`.

– *The following properties should hold at call time:*

`M` is currently instantiated to an atom. (term_typing:atom/1)

`F` is currently instantiated to an atom. (term_typing:atom/1)

`V` is a free variable. (term_typing:var/1)

set_menu_flag/3: PREDICATE

Usage: `set_menu_flag(M,F,V)`

– *Description:* Set the value `V` of the flag `F` in the menu (-branch) `M`.

– *The following properties should hold at call time:*

`M` is currently instantiated to an atom. (term_typing:atom/1)

`F` is currently instantiated to an atom. (term_typing:atom/1)

`V` is a free variable. (term_typing:var/1)

space/1: PREDICATE

Usage: `space(N)`

– *Description:* prints `N` spaces.

– *The following properties should hold at call time:*

`N` is a number. (basic_props:num/1)

get_menu_configs/1: PREDICATE

Usage: `get_menu_configs(X)`

– *Description:* Returns a list of atoms in `X` with the name of stored configurations.

– *The following properties should hold at call time:*

`X` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`X` is a list of `atom`s. (basic_props:list/2)

save_menu_config/1: PREDICATE

Usage: `save_menu_config(Name)`

– *Description:* Save the current flags configuration under the `Name` key.

– *The following properties should hold at call time:*

`Name` is an atom. (basic_props:atm/1)

remove_menu_config/1: PREDICATE

Usage: `remove_menu_config(Name)`

- *Description:* Remove the configuration stored with the `Name` key (the same provided in `save_menu_config/1`).
- *The following properties should hold at call time:*

  `Name` is an atom. (basic_props:atm/1)

restore_menu_config/1: PREDICATE

Usage: `restore_menu_config(Name)`

- *Description:* Restore the configuration saved with the `Name` key (the same provided in `save_menu_config/1`).
- *The following properties should hold at call time:*

  `Name` is an atom. (basic_props:atm/1)

show_menu_configs/0: PREDICATE

Usage:

- *Description:* Show all stored configurations.

show_menu_config/1: PREDICATE

Usage: `show_menu_config(C)`

- *Description:* Show specific configuration values pointed by `C` key (the same provided in `save_menu_config/1`).
- *The following properties should hold at call time:*

  `C` is an atom. (basic_props:atm/1)

get_menu_options/2: PREDICATE

Usage: `get_menu_options(Flag,V)`

- *Description:* Returns possilbe options in `V` by fail for the flag `Flag`.
- *The following properties should hold at call time:*

  `F` is currently instantiated to an atom. (term_typing:atom/1)

get_menu_flags/1: PREDICATE

Usage: `get_menu_flags(L)`

- *Description:* Return a list `L` of all current menu flags, composed by terms with the form (M,F,V), where M is the menu, F the flag, and V the value. This list can be used as argument of `restore_flags_list/1`
- *The following properties should hold at call time:*

  `L` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `L` is a list. (basic_props:list/1)

restore_menu_flags_list/1: PREDICATE

Usage: `restore_menu_flags_list(L)`

- *Description:* Restores menu flags. `L` is a list of tuple (M,F,V) where M is the menu, F is the flag, and V is the value of the flag F in the menu M.
- *The following properties should hold at call time:*

  `L` is a list. (basic_props:list/1)

get_menu_flags/2: PREDICATE

Usage: `get_menu_flags(M,L)`

- *Description:* Return a list `L` of the current menu `M` composed by terms with the form (F=V), F the flag, and V the value. This list can be used as argument of `restore_menu_flags/2`
- *The following properties should hold at call time:*

  `M` is any term. (basic_props:term/1)

  `L` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `L` is a list. (basic_props:list/1)

restore_menu_flags/2: PREDICATE

Usage: `restore_menu_flags(M,F)`

- *Description:* Restore the flag of the menu `M`. `F` is a list of terms F=V, which indicate the flag (F) and the value (V). `M` is the target menu to which those flags "belong". Additionally, `F` can contains terms like `changed_to_menu(NM)` that will put NM as the new target menu.
- *The following properties should hold at call time:*

  `M` is currently instantiated to an atom. (term_typing:atom/1)

  `F` is a list. (basic_props:list/1)

generate_js_menu/1: PREDICATE

Internal Info. Short description and general ideas about how the JS menu is generated.

The current model for the JS menu is an array of `menuq` (object in JS).

```
var assert_rtcheck = menus.length ;
var v_assert_rtcheck ;
menus[ menus.length ] = new menuq(
"assert_rtcheck",
        "Perform Run-Time Checks",
        "none,pred,pp_assrt,pp_code",
        "none",
        '((v_menu_level == "expert") &&
          (v_inter_all == "check_assertions"))' ) ;
```

A variable with the same name as the *flag name* is created with the value of the index of the menu in the array. Another variable with 'v_' (v = value) indicates the value (of the flag) choosed by the user in the combo-boxes that appear on the webpage (note you should not read this if you have not seen the webpage working). The object `menuq` holds

several things: the flag name (to find out the index in the array in some JS functions), the title, the options (notice there is no space in the options, this is important!), default option and the guard.

All the problem here is to generate the (JS) guard, because the rest of information is the same as in `menu_opt/6`. What code does is to execute the guard in prolog, and obtain a list of the form [flag=value,flag2=value2...]. Aditionally an element of the list can be another list, which indicates that the join operator (&& or ||) is swaped. For example: [a=1,[b=2,c=3],d=4] will be translated in JS like ((a==1)&&((b==2)||(c==3))&&(d==4)).

Prolog Guards (the ones in menu_opt) have been rewritten in order to do not generate free variables, i.e., they are finite guards now. So calling them with a variable in its first argument, we get the list like the named in the previous paragraph.

% Unfortunately, the current CiaoPP menu does not have all information % itself to concatenate several menus, i.e., when asking which kind of % action the user desire, depending on the answer one path or another % one is taken. Who decide which path? 'auto_inteface' does. We do have % to take this into account, because generating the assertions of a % "subpath" will produce the activation of it without permision of the % father (the menu which launch it).

Also, we have to keep in mind that menu has several submenus (or branches) defined and ones connect with others. For example, the menu

```
all, 'Select Menu Level'   # menu_level - naive.
all, 'Select Action Group' # inter_all  - analyze :: all_menu_branch.

check(1), 'Perform Compile-Time Checks' # assert_ctcheck - on.
...

ana     , 'Select Aliasing-Mode Analysis' # modes - shfr <- true.
```

defines 3 menus: `all`, `check(1)`, and `ana`. It usually happends that one menu invoques (connect or continue) with another menu (then it is a submenu or a branch). When generating the guards we have to add additional restrictions to the guards in order to make submenus do not appear in the incorrect moment. For example, `check(1)` menu will be only active if `menu_level = expert` and `inter_all=check_assertions` (more on this come later). How are several menus connected between each other? The process of connecting the menus is post-processing the selected flags (options) via post hook (the one defined after :: field). The post-processing hook only have to add the element `ask_menu(Branch,Level)` or `ask_menu(Branch)` to the selected flag list (the argument of the hook). So let us say that when calling all_menu_branch(X , Y) we get:

```
?- all_menu_branch(A,B).

A = [inter_all=optimize,menu_level=naive|_A],
B = [ask_menu(opt,0),inter_all=optimize,menu_level=naive|_A] ? ;

A = [inter_all=optimize,menu_level=expert|_A],
B = [ask_menu(opt,1),inter_all=optimize,menu_level=expert|_A] ? ;

A = [inter_all=analyze,menu_level=naive|_A],
B = [ask_menu(ana,0),inter_all=analyze,menu_level=naive|_A] ? ;

A = [inter_all=analyze,menu_level=expert|_A],
B = [ask_menu(ana,1),inter_all=analyze,menu_level=expert|_A] ? ;
```

```
A = [inter_all=check_assertions,menu_level=naive|_A],
B = [ask_menu(check,0),inter_all=check_assertions,menu_level=naive|_A] ? ;

A = [inter_all=check_assertions,menu_level=expert|_A],
B = [ask_menu(check,1),inter_all=check_assertions,menu_level=expert|_A] ? ;

A = [inter_all=check_certificate,menu_level=_B|_A],
B = [inter_all=check_certificate,menu_level=_B|_A] ? ;

A = [inter_all=optimize,_A,menu_level=naive|_B],
B = [ask_menu(opt,0),inter_all=optimize,_A,menu_level=naive|_B] ?
```

Notice that the last option contains free variables in the list (not taking the tail into account). That is the indicator for us to stop searching for more solutions. Additionally, we can have more complex things like:

```
?- opt_menu_branch(A,B).

A = [menu_level=naive,inter_optimize=_A|_B],
B = [ask_menu(_A,0),menu_level=naive,inter_optimize=_A|_B] ? ;

A = [menu_level=expert,inter_optimize=_A|_B],
B = [ask_menu(_A,1),menu_level=expert,inter_optimize=_A|_B] ? ;

A = [menu_level=naive,_B,inter_optimize=_A|_C],
B = [ask_menu(_A,0),menu_level=naive,_B,inter_optimize=_A|_C] ?
```

In this situation, the menu that will be asked will depend on the value of the flag `inter_optimize`, so we will have to generate as many ask_menu as possible values the flag has. The predicate `generate_menu_path/2` solves all this problem: for a given flag, it looks up to find out the guard composed by flags that will activate the flag. % This predicate also consider the problem of menu aliasing, % i.e., the option optimize in inter_ana menu launch opt menu, so % optimize is an alias for opt or vice-versa. For example, for the given flag `ass_not_stat_eval`, the path is: [v_menu_level=expert, v_inter_all=check_assertions], that means that `menu_level` flag has to have the value `"expert"` and the `inter_all` flag has to have the value `"check_assertions"` (note that menu_level and inter_all defines two menu branches). If we would execute only the precondition we would get: [v_assert_ctcheck=on].

There are % two one limitations imposed to the JS menu. % The menu level indicated by `menu_level` flag, is hirewired. The % other limitation is more serious. All flags values in the JS menu are mapped into one set of flags. In other words, changing a value of a shared flag by two menu branch will be reflected on the other branch. For example, changing type analysis in `analyze` menu branch, will modify the value of the same flag in `check assertions` branch.

The last point to name is about generated JS guards. This world is not fair, and sometimes happends things you just do not expect. Here is one of those things. When calling the Prolog guards with the list of current selected values, the cases that can occur are much less than when seeing all the possible combinations. Something like (a=true||a=1)&&(b=2), in Prolog guard execution (with instantiated things) will mean: the selected options are not nil, and a=1 and b=2, or in other words: (a==1)&&(b==2). But in JS, unfortunately means: b=2. Nowadays, this problem is solved by `clean_imperative_guard/2` which tries to remove all stupid `true` conditions, but I am sure Murphy is listening to me now and he started to create an user to make the call `clean_imperative_guard/2` generates wrong answer.

Usage: `generate_js_menu(DoNotIncludeList)`

– *Description:* Reads all multifile `menu_opt/6` predicates and writes in default output a JavaScript Menu.
– *The following properties should hold at call time:*
`DoNotIncludeList` is a list. (basic_props:list/1)

eq/3: PREDICATE

Usage: `eq(Type,A,B)`

– *Description:* `Type` is the value returned by the 2nd arg of `uni_type`. `A` and `B` are whatever terms. This predicate success if they are equal (like A=B).

neq/3: PREDICATE

Usage: `neq(Type,A,B)`

– *Description:* `Type` is the value returned by the 2nd arg of `uni_type`. `A` and `B` are whatever terms. The semantic is similar to A == B.

uni_type/2: PREDICATE

Usage: `uni_type(Var,Type)`

– *Description:* `Var` should be the argument passed to the menu guard. `Type` is an abstract type that decides how unifications should be done in `eq/3` and `neq/3`.

vmember/2: PREDICATE

Usage: `vmember(Var,List)`

– *Description:* It is `member` equivalent predicate to be used in guards.

menu_flag_values/1: REGTYPE

Usage: `menu_flag_values(X)`

– *Description:* Flag values

127.3 Documentation on multifiles (`menu_generator`)

$is_persistent/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/2: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/4: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

menu_default/3: PREDICATE

The predicate is *multifile*.

Usage 1: `menu_default(Menu,Flag,DefaultValue)`

– *Description:* `Menu` is a term that has to correspond with the 1st argument of `Menu`. `Flag` is the desired flag to have a default value. `DefaultValue` is the default value of `Flag`.

– *The following properties should hold at call time:*

`Menu` is any term. (basic_props:term/1)

`Flag` is an atom. (basic_props:atm/1)

`DefaultValue` is an atom. (basic_props:atm/1)

Usage 2: `menu_default(Menu,Flag,DefaultValue)`

– *The following properties hold upon exit:*

`Menu` is an atom. (basic_props:atm/1)

`Flag` is an atom. (basic_props:atm/1)

`DefaultValue` is an atom. (basic_props:atm/1)

Usage 3: `menu_default(Menu,Flag,DefaultValue)`

– *Description:* This call mode can be used to ask which flags and its values has a menu `menu`

– *The following properties should hold at call time:*

`Menu` is an atom. (basic_props:atm/1)

`Flag` is a free variable. (term_typing:var/1)

`DefaultValue` is a free variable. (term_typing:var/1)

Usage 4: `menu_default(Menu,Flag,DefaultValue)`

– *Description:* This call mode can be used to ask which value have the flag `Flag` in the menu `menu`

– *The following properties should hold at call time:*

`Menu` is an atom. (basic_props:atm/1)

`Flag` is an atom. (basic_props:atm/1)

`DefaultValue` is a free variable. (term_typing:var/1)

menu_opt/6: PREDICATE

The predicate is *multifile*.

Usage 1: `menu_opt(Menu,Flag,Text,Guard,BeforePrinting,SelectedHook)`

– *Description:* `Menu` is a term that specifies the menu name. It can be an atom or just a predicate of arity 1, where the 1st argument indicates the menu level (i.e., ana(1) is the level 1 of 'ana' menu). `Flag` is the flag that will be asked.

`Text` is the test that will be printed when asking the `Flag`.

`Guard` is a predicate of arity 1 that is invoked to see if the flag should be asked. The argument is the selected menu options till moment in the way: [flag1=value1, flag2=value2, ...].

`BeforePrinting` is a predicate of arity 0, that is invoked whenever the menu option has been selected the validator menu options chooser.

`SelectedHook` is a predicate of arity 2, that is invoked whenever the flag has been selected by the user. The 1st argument are the current selected values, including the current flag, and in the 2nd argument the possible modified list is expected.

In summary, if `Guard` holds, then `BeforePrinting` is executed (no action is taken whether it fails or not), and after the user has types the option `SelectedHook` is invoked.

– *The following properties should hold at call time:*

`Menu` is any term. (basic_props:term/1)

`Flag` is an atom. (basic_props:atm/1)

`Text` is an atom. (basic_props:atm/1)

`Guard` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`BeforePrinting` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`SelectedHook` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 2: `menu_opt(Menu,Flag,Text,Guard,BeforePrinting,SelectedHook)`

– *The following properties should hold at call time:*

`Menu` is any term. (basic_props:term/1)

`Flag` is any term. (basic_props:term/1)

`Text` is any term. (basic_props:term/1)

`Guard` is any term. (basic_props:term/1)

`BeforePrinting` is any term. (basic_props:term/1)

`SelectedHook` is any term. (basic_props:term/1)

hook_menu_flag_values/3: PREDICATE

The predicate is *multifile*.

Usage: `hook_menu_flag_values(Menu,Flag,Values)`

– *Description:* It is a hook. It is invoked whenever a menu question is printed. `Values` is a term which specifies the possible values. If `Values` is alist(List) -atom list-, then menu will check if the typed value by user belongs to List. If `Values` is a term ask(T,Flag), the menu will invoke `hook_menu_check_flag_value/3` hook to check if introduced value is valid.

– *The following properties should hold at call time:*

`Menu` is currently instantiated to an atom. (term_typing:atom/1)

`Flag` is currently instantiated to an atom. (term_typing:atom/1)

`Values` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

Flag values (menu_generator:menu_flag_values/1)

hook_menu_check_flag_value/3: PREDICATE

The predicate is *multifile*.

Usage: `hook_menu_check_flag_value(M,F,V)`

– *Description:* It is a hook. It is invoked whenever the menu needs to check whether the answer introduced for the menu `M` is correct. This happens when `hook_menu_flag_values/3` returns in its second argument something different than alist(_).

hook_menu_flag_help/3: PREDICATE

The predicate is *multifile*.

Usage: `hook_menu_flag_help(M,F,H)`

– *Description:* It is a hook. It is invoked whenever the user ask for a help description, `H`, of the flag `F` in the menu `M`.

hook_menu_default_option/3: PREDICATE

The predicate is *multifile*.

Usage: `hook_menu_default_option(M,F,D)`

– *Description:* It is a hook. It is invoked whenever the menu needs to offer a default option to the user in the menu `M` and it has not been neither introduced before nor specified by `menu_default/3`.

127.4 Known bugs and planned improvements (`menu_generator`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

128 Interface to daVinci

Author(s): Francisco Bueno.

This library allows connecting a Ciao Prolog application with daVinci V2.X.

The communication is based on a two-way channel: after daVinci is started, messages are sent in to it and read in from it on demand by different Prolog predicates. Messages are sent via writing the term as text; messages are received by reading text and returning an atom. Commands sent and answers received are treated as terms from the Prolog side, since for daVinci they are text but have term syntax; the only difficulty lies in strings, for which special Prolog syntax is provided.

See accompanying file `library(davinci(commands))` for examples on the use of this library.

daVinci is developed by U. of Bremen, Germany.

128.1 Usage and interface (`davinci`)

- **Library usage:**
  `:- use_module(library(davinci)).`
- **Exports:**
  - *Predicates:*
    `davinci/0`, `topd/0`, `davinci_get/1`, `davinci_get_all/1`, `davinci_put/1`, `davinci_quit/0`, `davinci_ugraph/1`, `davinci_lgraph/1`, `ugraph2term/2`, `formatting/2`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `prompt`, `format`, `read`, `graphs/ugraphs`, `write`, `system`.

128.2 Documentation on exports (`davinci`)

davinci/0: PREDICATE

Start up a daVinci process.

topd/0: PREDICATE

A toplevel to send to daVinci commands from standard input.

davinci_get/1: PREDICATE

Usage: `davinci_get(Term)`

- *Description:* Get a message from daVinci. `Term` is a term corresponding to daVinci's message.

davinci_get_all/1: PREDICATE

Usage: `davinci_get_all(List)`

- *Description:* Get all pending messages. `List` is a list of terms as in `davinci_get/1`.

– *The following properties should hold upon exit:*
`List` is a list. (basic_props:list/1)

davinci_put/1: PREDICATE

Usage: `davinci_put(Term)`

– *Description:* Send a command to daVinci.
– *The following properties should hold at call time:*
davinci:davinci_command(Term) (davinci:davinci_command/1)

davinci_quit/0: PREDICATE

Exit daVinci process. All pending answers are lost!

davinci_ugraph/1: PREDICATE

Usage: `davinci_ugraph(Graph)`

– *Description:* Send a graph to daVinci.
– *The following properties should hold at call time:*
davinci:ugraph(Graph) (davinci:ugraph/1)

davinci_lgraph/1: PREDICATE

Usage: `davinci_lgraph(Graph)`

– *Description:* Send a labeled graph to daVinci.
– *The following properties should hold at call time:*
davinci:lgraph(Graph) (davinci:lgraph/1)

ugraph2term/2: PREDICATE

No further documentation available for this predicate.

formatting/2: PREDICATE

No further documentation available for this predicate.

128.3 Documentation on internals (`davinci`)

davinci_command/1: PROPERTY

Syntactically, a command is a term. Semantically, it has to correspond to a command understood by daVinci. Two terms are interpreted in a special way: `string/1` and `text/1`: `string(Term)` is given to daVinci as `"Term"`; `text(List)` is given as `"Term1 Term2 ...Term "` for each `Term` in `List`. If your term has functors `string/1` and `text/1` that you don't want to be interpreted this way, use it twice, i.e., `string(string(Term))` is given to daVinci as `string(Term')` where `Term'` is the interpretation of `Term`.

ugraph/1: PROPERTY

`ugraph(Graph)`

`Graph` is a term which denotes an ugraph as in `library(ugraphs)`. Vertices of the form `node/2` are interpreted in a special way: `node(Term,List)` is interpreted as a vertex `Term` with attributes `List`. `List` is a list of terms conforming the syntax of `davinci_put/1` and corresponding to daVinci's graph nodes attributes. If your vertex has functor `node/2` and you don't want it to be interpreted this way, use it twice, i.e., `node(node(T1,T2),[])` is given to daVinci as vertex `node(T1,T2)`. A vertex is used both as label and name of daVinci's graph node. daVinci's graph edges have label `V1-V2` where `V1` is the source and `V2` the sink of the edge. There is no support for multiple edges between the same two vertices.

lgraph/1: PROPERTY

`lgraph(Graph)`

`Graph` is a term which denotes a wgraph as in `library(wgraphs)`, except that the weights are labels, i.e., they do not need to be integers. Vertices of the form `node/2` are interpreted in a special way. Edge labels are converted into special intermediate vertices. Duplicated labels are solved by adding dummy atoms `''`. There is no support for multiple edges between the same two vertices.

129 The Tcl/Tk interface

Author(s): Montse Iglesias Urraca, The CLIP Group.

The `tcltk` library package is a bidirectional interface to the *Tcl* language and the *Tk* toolkit. Tcl is an interpreted scripting language with many extension packages, particularly the graphical interface toolkit, Tk. The interaction between both languages is expressed in terms of an interface between the Tcl/Tk process and the Prolog process. This approach allows the development of mixed applications where both sides, Tcl/Tk and Prolog, can be combined in order to exploit their respective capabilities.

This library uses two sockets to connect both the Tcl and the Prolog processes: *event_socket* and *term_socket*. There are also two Tcl global variables: *prolog_variables* and *terms*. The value of any of the bound variables in a goal will be stored in the array `prolog_variables` with the variable name as index. *Terms* is the string which contains the printed representation of prolog *terms*.

Prolog to Tcl

The Tcl/Tk side waits for requests from the Prolog side, and executes the Tcl/Tk code received. Also, the Tcl/Tk side handles the events and exceptions which may be raised on its side, passing on control to the Prolog side in case it is necessary.

To use Tcl, you must create a *Tcl interpreter* object and send commands to it. A *Tcl* command is specified as follows:

```
Command         --> Atom  { other than [] }
                  | Number
                  | chars(PrologString)
                  | write(Term)
                  | format(Fmt,Args)
                  | dq(Command)
                  | br(Command)
                  | sqb(Command)
                  | min(Command)
                  | ListOfCommands
ListOfCommands  --> []
                  |[Command|ListOfCommands]
```

where:

`Atom` denotes the printed representation of the atom.

`Number` denotes their printed representations.

`chars(PrologString)`
: denotes the string represented by *PrologString* (a list of character codes).

`write(Term)`
: denotes the string that is printed by the corresponding built-in predicate.

`format(Term)`
: denotes the string that is printed by the corresponding built-in predicate.

`dq(Command)`
: denotes the string specified by *Command*, enclosed in double quotes.

`br(Command)`
: denotes the string specified by *Command*, enclosed in braces.

`sqb(Command)`
: denotes the string specified by *Command*, enclosed in square brackets.

`min(Command)`
: denotes the string specified by *Command*, immediately preceded by a hyphen.

`ListOfCommands`
: denotes the strings denoted by each element, separated by spaces.

The predicates to use Tcl from Prolog are `tcl_new/1`, `tcl_delete/1`, `tcl_eval/3`, and `tcl_event/3`.

An example of use with Prolog as master and Tcl as slave, consisting of a GUI to a program which calculates the factorial of a number:

```
:- use_module(library(tcltk)).

go :-
        tcl_new(X),
        tcl_eval(X,[button,'.b',min(text),dq('Compute!')],_),
        tcl_eval(X,[button,'.c','-text',dq('Quit')],_),
        tcl_eval(X,[entry,'.e1',min(textvariable),'inputval'],_),
        tcl_eval(X,[label,'.l1',min(text),dq('The factorial of ')],_),
        tcl_eval(X,[pack, '.l1','.e1'],_),
        tcl_eval(X,[entry,'.e2',min(textvariable),'outputval'],_),
        tcl_eval(X,[label,'.l2',min(text),dq('is  ')],_),
        tcl_eval(X,[pack, '.l2','.e2'],_),
        tcl_eval(X,[pack,'.b','.c',min(side),'left'],_),
        tcl_eval(X,[bind,'.b','<ButtonPress-1>',
                br([set,'inputval','$inputval','\n',
                prolog_one_event,
                dq(write(execute(tk_test_aux:factorial('$inputval', 'Outputval')))),
                '\n',
                set, 'outputval','$prolog_variables(Outputval)'])],
                _),
        tcl_eval(X,[bind,'.c','<ButtonPress-1>',
                br([prolog_one_event,
                dq(write(execute(exit_tk_event_loop)))])],
                _),
        tk_event_loop(X).
```

Tcl to Prolog

This is the usual way to build a GUI application. The slave, Prolog, behaves as a server that fulfills eventual requests from the master side, Tcl. At some point, during the user interaction with the GUI, an action may take place that triggers the execution of some procedure on the slave side (a form submit, for example). Thus, the slave is invoked, performs a service, and returns the result to the GUI through the socket connection.

This library includes two main specific Tcl commands:

`prolog` *Goal*
: *Goal* is a string containing the printed representation of a Prolog goal. The goal will be called in the user module unless it is prefixed with another module name. The call is always deterministic and its can be either of the following:

  1, in case of success
  : The value of any of the variables in the goal that is bound to a term will be returned to Tcl in the array prolog_variables with the variable name as index.

0, if the execution fails
: The Prolog exception Tcl exception is raised. The error message will be "Prolog Exception: " appended with a string representation of such exception.

`prolog_event` *Term*
: Adds the new *term* to the *terms* queue. These can be later retrieved through predicates `tcl_event/3` and `tk_next_event/2`.

Additionally, seven extra Tcl commands are defined.

`prolog_delete_event`
: Deletes the first *term* of the *terms* queue.

`prolog_list_events`
: Sends all the *terms* of the *terms* queue through the *event_socket*. The last element is *end_of_event_list*.

`prolog_cmd` *Command*
: Receives as an argument the Tcl/Tk code, evaluates it and returns through the *term_socket* the term *tcl_error* in case of error or the term *tcl_result* with the result of the command executed. If the command is *prolog*, upon return, the goal run on the prolog side is received. In order to get the value of the variables, predicates are compared using the *unify_term* command. Returns 0 when the sript runs without errors, and 1 if there is an error.

`prolog_one_event` *Term*
: Receives as an argument the *term* associated to one of the Tk events. Sends the *term* through the *event_socket* and waits for its unification. Then *unify_term* command is called to update the *prolog_variables* array.

`prolog_thread_event` *Term*
: Receives as an argument the *term* associated to one of the Tk events. Sends the *term* through the *event_socket* and waits for its unification. Then *unify_term* command is called to update the *prolog_variables* array. In this case the *term_socket* is non blocking.

`convert_variables` *String*
: Its argument is a string containing symbols that can not be sent through the sockets. This procedure deletes them from the input string and returns the new string.

`unify_term` *Term1 Term2*
: Unifies *Term1* and *Term2* and updates the the *prolog_variables* array.

The predicates to use Prolog from Tcl are `tk_event_loop/1`, `tk_main_loop/1`, `tk_new/2`, and `tk_next_event/2`.

An example of use with Tcl as master and Prolog as slave, implementing the well known "Hello, world!" dummy program (more can be seen in directory examples):

Prolog side:

```
:- use_module(library(tcltk)).
:- use_package(classic).

hello('Hello, world!').

go :-
        tk_new([name('Simple')], Tcl),
        tcl_eval(Tcl, 'source simple.tcl', _),
        tk_main_loop(Tcl),
```

```
        tcl_delete(Tcl).
```

Tcl side (simple.tcl):

```
label .l -textvariable tvar
button .b -text "Go!" -command {run}
pack .l .b -side top

proc run {} {

    global prolog_variables
    global tvar

    prolog hello(X)
    set tvar $prolog_variables(X)
}
```

129.1 Usage and interface (tcltk)

- **Library usage:**
  `:- use_module(library(tcltk)).`
- **Exports:**
  - *Predicates:*
    `tcl_new/1`, `tcl_eval/3`, `tcl_delete/1`, `tcl_event/3`, `tk_event_loop/1`, `tk_main_loop/1`, `tk_new/2`, `tk_next_event/2`.
  - *Regular Types:*
    `tclInterpreter/1`, `tclCommand/1`.
- **Other modules used:**
  - *System library modules:*
    `tcltk/tcltk_low_level`, `write`.

129.2 Documentation on exports (tcltk)

tcl_new/1: PREDICATE

Usage: `tcl_new(TclInterpreter)`

- *Description:* Creates a new interpreter, initializes it, and returns a handle to it in `TclInterpreter`.
- *Call and exit should be compatible with:*
  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk:tclInterpreter/1)
- *The following properties should hold at call time:*
  `TclInterpreter` is a free variable. (term_typing:var/1)

tcl_eval/3: PREDICATE

Meta-predicate with arguments: `tcl_eval(?,?,addmodule(?))`.

Usage: `tcl_eval(TclInterpreter,Command,Result)`

- *Description:* Evaluates the commands given in `Command` in the Tcl interpreter `TclInterpreter`. The result will be stored as a string in `Result`. If there is an error in *Command* an exception is raised. The error messages will be *Tcl Exception:* if the error is in the syntax of the Tcl/Tk code or *Prolog Exception:*, if the error is in the prolog term.
- *Call and exit should be compatible with:*

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk:tclInterpreter/1)

  `Command` is a *Tcl* command. (tcltk:tclCommand/1)

  `Result` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*

  `TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Command` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Result` is a free variable. (term_typing:var/1)

tcl_delete/1: PREDICATE

Usage: `tcl_delete(TclInterpreter)`

- *Description:* Given a handle to a Tcl interpreter in variable `TclInterpreter`, it deletes the interpreter from the system.
- *Call and exit should be compatible with:*

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk:tclInterpreter/1)
- *The following properties should hold at call time:*

  `TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

tcl_event/3: PREDICATE

Usage: `tcl_event(TclInterpreter,Command,Events)`

- *Description:* Evaluates the commands given in `Command` in the Tcl interpreter whose handle is provided in `TclInterpreter`. `Events` is a list of terms stored from Tcl by *prolog_event*. Blocks until there is something on the event queue
- *Call and exit should be compatible with:*

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk:tclInterpreter/1)

  `Command` is a *Tcl* command. (tcltk:tclCommand/1)

  `Events` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*

  `TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Command` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Events` is a free variable. (term_typing:var/1)

tclInterpreter/1: REGTYPE

Usage: `tclInterpreter(I)`

– *Description:* `I` is a reference to a *Tcl* interpreter.

tclCommand/1: REGTYPE

Usage: `tclCommand(C)`

– *Description:* `C` is a *Tcl* command.

tk_event_loop/1: PREDICATE

Usage: `tk_event_loop(TclInterpreter)`

– *Description:* Waits for an event and executes the goal associated to it. Events are stored from Tcl with the *prolog* command. The unified term is sent to the Tcl interpreter in order to obtain the value of the tcl array of *prolog_variables*. If the term received does not have the form `execute(Goal)`, the predicate silently exits. If the execution of `Goal` raises a Prolog error, the interpreter is deleted and an error message is given.

– *Call and exit should be compatible with:*

`TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk:tclInterpreter/1)

– *The following properties should hold at call time:*

`TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

tk_main_loop/1: PREDICATE

Usage: `tk_main_loop(TclInterpreter)`

– *Description:* Passes control to Tk until all windows are gone.

– *Call and exit should be compatible with:*

`TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk:tclInterpreter/1)

– *The following properties should hold at call time:*

`TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

tk_new/2: PREDICATE

Usage: `tk_new(Options,TclInterpreter)`

– *Description:* Performs basic Tcl and Tk initialization and creates the main window of a Tk application.`Options` is a list of optional elements according to:

`name(+ApplicationName)`
: Sets the Tk main window title to `ApplicationName`. It is also used for communicating between Tcl/Tk applications via the Tcl *send* command. Default name is an empty string.

`display(+Display)`
: Gives the name of the screen on which to create the main window. Default is normally determined by the `DISPLAY` environment variable.

`file` Opens the sript `file`. Commands will not be read from standard input and the execution returns back to Prolog only after all windows (and the interpreter) have been deleted.

– *Call and exit should be compatible with:*

`Options` is a list. (basic_props:list/1)

`TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk:tclInterpreter/1)

– *The following properties should hold at call time:*

`Options` is currently a term which is not a free variable. (term_typing:nonvar/1)

`TclInterpreter` is a free variable. (term_typing:var/1)

tk_next_event/2: PREDICATE

Usage: `tk_next_event(TclInterpreter,Event)`

– *Description:* Processes events until there is at least one Prolog event associated with `TclInterpreter`. `Event` is the term correspondig to the head of a queue of events stored from Tcl with the *prolog_event* command.

– *Call and exit should be compatible with:*

`TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk:tclInterpreter/1)

`Event` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold at call time:*

`TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Event` is a free variable. (term_typing:var/1)

130 Low level interface library to Tcl/Tk

Author(s): Montse Iglesias Urraca.

The `tcltk_low_level` library defines the low level interface used by the `tcltk` library. Essentially it includes all the code related directly to the handling of sockets and processes. This library should normally not be used directly by user programs, which use `tcltk` instead. On the other hand in some cases it may be useful to undertand how this library works in order to understand possible problems in programs that use the `tcltk` library.

130.1 Usage and interface (`tcltk_low_level`)

- **Library usage:**
  `:- use_module(library(tcltk_low_level)).`
- **Exports:**
  - *Predicates:*
    `new_interp/1`, `new_interp/2`, `new_interp_file/2`, `tcltk/2`, `tcltk_raw_code/2`, `receive_result/2`, `send_term/2`, `receive_event/2`, `receive_list/2`, `receive_confirm/2`, `delete/1`.
- **Other modules used:**
  - *System library modules:*
    `terms`, `sockets/sockets`, `system`, `write`, `read`, `strings`, `format`.

130.2 Documentation on exports (`tcltk_low_level`)

new_interp/1: PREDICATE

Usage: `new_interp(TclInterpreter)`

- *Description:* Creates two sockets to connect to the *wish* process, the term socket and the event socket, and opens a pipe to process *wish* in a new shell.
- *Call and exit should be compatible with:*
  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)
- *The following properties should hold at call time:*
  `TclInterpreter` is a free variable. (term_typing:var/1)

new_interp/2: PREDICATE

Usage: `new_interp(TclInterpreter,Options)`

- *Description:* Creates two sockets, the term socket and the event socket, and opens a pipe to process *wish* in a new shell invoked with the `Options`.
- *Call and exit should be compatible with:*
  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)
  `Options` is currently instantiated to an atom. (term_typing:atom/1)
- *The following properties should hold at call time:*
  `TclInterpreter` is a free variable. (term_typing:var/1)
  `Options` is currently a term which is not a free variable. (term_typing:nonvar/1)

new_interp_file/2: PREDICATE

Usage: `new_interp_file(FileName,TclInterpreter)`

- *Description:* Creates two sockets, the term socket and the event socket, and opens a pipe to process *wish* in a new shell invoked with a `FileName`. `FileName` is treated as a name of a sript file
- *Call and exit should be compatible with:*

  `FileName` is a string (a list of character codes). (basic_props:string/1)

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `TclInterpreter` is a free variable. (term_typing:var/1)

tcltk/2: PREDICATE

Usage: `tcltk(Code,TclInterpreter)`

- *Description:* Sends the `Code` converted to string to the `TclInterpreter`
- *Call and exit should be compatible with:*

  `Code` is a *Tcl* command. (tcltk_low_level:tclCommand/1)

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)
- *The following properties should hold at call time:*

  `Code` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

tcltk_raw_code/2: PREDICATE

Usage: `tcltk_raw_code(String,TclInterpreter)`

- *Description:* Sends the tcltk code items of the `Stream` to the `TclInterpreter`
- *Call and exit should be compatible with:*

  `String` is a string (a list of character codes). (basic_props:string/1)

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)
- *The following properties should hold at call time:*

  `String` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

receive_result/2: PREDICATE

Usage: `receive_result(Result,TclInterpreter)`

- *Description:* Receives the `Result` of the last *TclCommand* into the `TclInterpreter`. If the *TclCommand* is not correct the *wish* process is terminated and a message appears showing the error
- *Call and exit should be compatible with:*

  `Result` is a string (a list of character codes). (basic_props:string/1)

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)

– *The following properties should hold at call time:*

`Result` is a free variable. (term_typing:var/1)

`TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

send_term/2: PREDICATE

Usage: `send_term(String,TclInterpreter)`

– *Description:* Sends the goal executed to the `TclInterpreter`. `String` has the predicate with unified variables

– *Call and exit should be compatible with:*

`String` is a string (a list of character codes). (basic_props:string/1)

`TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)

– *The following properties should hold at call time:*

`String` is currently a term which is not a free variable. (term_typing:nonvar/1)

`TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

receive_event/2: PREDICATE

Usage: `receive_event(Event,TclInterpreter)`

– *Description:* Receives the `Event` from the event socket of the `TclInterpreter`.

– *Call and exit should be compatible with:*

`Event` is a list. (basic_props:list/1)

`TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)

– *The following properties should hold at call time:*

`Event` is a free variable. (term_typing:var/1)

`TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

receive_list/2: PREDICATE

Usage: `receive_list(List,TclInterpreter)`

– *Description:* Receives the `List` from the event socket of the `TclInterpreter`.The `List` has all the predicates that have been inserted from Tcl/Tk with the command prolog_event. It is a list of terms.

– *Call and exit should be compatible with:*

`List` is a list. (basic_props:list/1)

`TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)

– *The following properties should hold at call time:*

`List` is a free variable. (term_typing:var/1)

`TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

receive_confirm/2: PREDICATE

Usage: `receive_confirm(String,TclInterpreter)`

- *Description:* Receives the `String` from the event socket of the `TclInterpreter` when a term inserted into the event queue is managed.
- *Call and exit should be compatible with:*

  `String` is a string (a list of character codes). (basic_props:string/1)

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)
- *The following properties should hold at call time:*

  `String` is a free variable. (term_typing:var/1)

  `TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

delete/1: PREDICATE

Usage: `delete(TclInterpreter)`

- *Description:* Terminates the *wish* process and closes the pipe, term socket and event socket. Deletes the interpreter `TclInterpreter` from the system
- *Call and exit should be compatible with:*

  `TclInterpreter` is a reference to a *Tcl* interpreter. (tcltk_low_level:tclInterpreter/1)
- *The following properties should hold at call time:*

  `TclInterpreter` is currently a term which is not a free variable. (term_typing:nonvar/1)

130.3 Documentation on internals (`tcltk_low_level`)

core/1: PREDICATE

Usage: `core(String)`

- *Description:* `core/1` is a set of facts which contain `String`s to be sent to the Tcl/Tk interpreter on startup. They implement miscelaneous Tcl/Tk procedures which are used by the Tcl/Tk interface.
- *Call and exit should be compatible with:*

  `String` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*

  `String` is a free variable. (term_typing:var/1)

131 The PiLLoW Web programming library

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This package implements the PiLLoW library [CHV96a]. The following three chapters document, respectively, the predicates for HTML/XML/CGI programming, the predicate for HTTP conectivity, and the types used in the definition of the predicates (key for fully understanding the other predicates). You can find a paper and some additional information in the `library/pillow/doc` directory of the distribution, and in the WWW at `http://clip.dia.fi.upm.es/Software/pillow/pillow.html`. There is also a *PiLLoW on-line tutorial* (slides) at `http://clip.dia.fi.upm.es/logalg/slides/C_pillow/C_pillow.html` which illustrates the basic features and provides a number of examples of PiLLoW use.

131.1 Installing PiLLoW

To correctly install PiLLoW, first, make sure you downloaded the right version of PiLLoW (there are different versions for different LP/CLP systems; the version that comes with Ciao is of course the right one for Ciao). Then, please follow these steps:

1. Copy the files in the `images` directory to a WWW accessible directory in your server.
2. Edit the file `icon_address.pl` and change the fact to point to the URL to be used to access the images above.
3. In the Ciao system the files are in the correct place, in other systems copy the files `pillow.pl` and `icon_address.pl` to a suitable directory so that your Prolog system will find them.

131.2 Usage and interface (`pillow_doc`)

- **Library usage:**
  `:- use_package(pillow).`
  or
  `:- module(...,...,[pillow]).`
- **New operators defined:**
  `$/2` [150,xfx], `$/1` [150,fx].
- **Other modules used:**
  - *System library modules:*
    `pillow/http`, `pillow/html`.

132 HTML/XML/CGI programming

Author(s): Daniel Cabeza, Manuel Hermenegildo, Sacha Varma.

This module implements the predicates of the PiLLoW package related to HTML/ XML generation and parsing, CGI and form handlers programming, and in general all the predicates which do not imply the use of the HTTP protocol.

132.1 Usage and interface (`html`)

- **Library usage:**
  `:- use_module(library(html)).`
- **Exports:**
  - *Predicates:*
    `output_html/1`, `html2terms/2`, `xml2terms/2`, `html_template/3`, `html_report_error/1`, `get_form_input/1`, `get_form_value/3`, `form_empty_value/1`, `form_default/3`, `set_cookie/2`, `get_cookies/1`, `url_query/2`, `url_query_amp/2`, `url_query_values/2`, `my_url/1`, `url_info/2`, `url_info_relative/3`, `form_request_method/1`, `icon_address/2`, `html_protect/1`, `http_lines/3`.
  - *Multifiles:*
    `define_flag/3`, `html_expansion/2`.
- **Other modules used:**
  - *System library modules:*
    `strings`, `lists`, `system`, `read`, `pillow/pillow_aux`, `pillow/pillow_types`.

132.2 Documentation on exports (`html`)

output_html/1: PREDICATE

`output_html(HTMLTerm)`

Outputs `HTMLTerm`, interpreted as an `html_term/1`, to current output stream.

Usage:

- *The following properties should hold at call time:*
  `HTMLTerm` is a term representing HTML code. (pillow_types:html_term/1)

html2terms/2: PREDICATE

`html2terms(String,Terms)`

`String` is a character list containing HTML code and `Terms` is its prolog structured representation.

Usage 1:

- *Description:* Translates an HTML-term into the HTML code it represents.
- *The following properties should hold at call time:*
  `String` is a free variable. (term_typing:var/1)
  `Terms` is a term representing HTML code. (pillow_types:html_term/1)

– *The following properties hold upon exit:*

`String` is a string (a list of character codes). (basic_props:string/1)

Usage 2:

– *Description:* Translates HTML code into a structured HTML-term.

– *Calls should, and exit will be compatible with:*

`Terms` is a term representing HTML code in canonical form. (pillow_types:canonic_html_term/1)

– *The following properties should hold at call time:*

`String` is a string (a list of character codes). (basic_props:string/1)

– *The following properties hold upon exit:*

`Terms` is a term representing HTML code in canonical form. (pillow_types:canonic_html_term/1)

xml2terms/2: PREDICATE

`xml2terms(String,Terms)`

`String` is a character list containing XML code and `Terms` is its prolog structured representation.

Usage 1:

– *Description:* Translates a XML-term into the XML code it represents.

– *The following properties should hold at call time:*

`String` is a free variable. (term_typing:var/1)

`Terms` is a term representing HTML code. (pillow_types:html_term/1)

– *The following properties hold upon exit:*

`String` is a string (a list of character codes). (basic_props:string/1)

Usage 2:

– *Description:* Translates XML code into a structured XML-term.

– *Calls should, and exit will be compatible with:*

`Terms` is a term representing XML code in canonical form. (pillow_types:canonic_xml_term/1)

– *The following properties should hold at call time:*

`String` is a string (a list of character codes). (basic_props:string/1)

– *The following properties hold upon exit:*

`Terms` is a term representing XML code in canonical form. (pillow_types:canonic_xml_term/1)

html_template/3: PREDICATE

`html_template(Chars,Terms,Dict)`

Interprets `Chars` as an HTML template returning in `Terms` the corresponding structured HTML-term, which includes variables, and unifying `Dict` with a dictionary of those variables (an incomplete list of *name*=*Var* pairs). An HTML template is standard HTML code, but in which "slots" can be defined and given an identifier. These slots represent parts of the HTML code in which other HTML code can be inserted, and are represented in the HTML-term as free variables. There are two kinds of variables in templates:

- Variables representing page contents. A variable with name *name* is defined with the special tag `<V>`*name*`</V>`.
- Variables representing tag attributes. They occur as an attribute or an attribute value starting with `_`, followed by its name, which must be formed by alphabetic characters.

As an example, suposse the following HTML template:

```
<html>
<body bgcolor=_bgcolor>
<v>content</v>
</body>
</html>
```

The following query in the Ciao toplevel shows how the template is parsed, and the dictionary returned:

```
?- file_to_string('template.html',_S), html_template(_S,Terms,Dict).

Dict = [bgcolor=_A,content=_B|_],
Terms = [env(html,[],["
",env(body,[bgcolor=_A],["
",_B,"
"]),"
"]),"
"] ?

yes
```

If a dictionary with values is supplied at call time, then variables are unified accordingly inside the template:

```
?- file_to_string('template.html',_S),
   html_template(_S,Terms,[content=b("hello world!"),bgcolor="white"]).

Terms = [env(html,[],["
",env(body,[bgcolor="white"],["
",b("hello world!"),"
"]),"
"]),"
"] ?

yes
```

Usage:

– *Calls should, and exit will be compatible with:*

`Terms` is a term representing HTML code in canonical form. (pillow_types:canonic_html_term/1)

`Dict` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Chars` is a string (a list of character codes). (basic_props:string/1)

– *The following properties hold upon exit:*

`Terms` is a term representing HTML code in canonical form. (pillow_types:canonic_html_term/1)

`Dict` is a list. (basic_props:list/1)

html_report_error/1: PREDICATE

Usage: `html_report_error(Error)`

– *Description:* Outputs error `Error` as a standard HTML page.

get_form_input/1: PREDICATE

`get_form_input(Dict)`

Translates input from the form (with either the POST or GET methods, and even with CONTENT_TYPE multipart/form-data) to a dictionary `Dict` of *attribute=value* pairs. If the flag `raw_form_values` is `off` (which is the default state), it translates empty values (which indicate only the presence of an attribute) to the atom `'$empty'`, values with more than one line (from text areas or files) to a list of lines as strings, the rest to atoms or numbers (using `name/2`). If the flag `on`, it gives all values as atoms, without translations.

Usage:

– *The following properties should hold at call time:*

`Dict` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`Dict` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (pillow_types:form_dict/1)

get_form_value/3: PREDICATE

`get_form_value(Dict,Var,Val)`

Unifies `Val` with the value for attribute `Var` in dictionary `Dict`. Does not fail: value is `''` if not found (this simplifies the programming of form handlers when they can be accessed directly).

Usage:

– *Calls should, and exit will be compatible with:*

`Val` is a value of an attribute of a form. (pillow_types:form_value/1)

– *The following properties should hold at call time:*

`Dict` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (pillow_types:form_dict/1)

`Var` is an atom. (basic_props:atm/1)

– *The following properties hold upon exit:*

`Val` is a value of an attribute of a form. (pillow_types:form_value/1)

form_empty_value/1: PREDICATE

Usage: `form_empty_value(Term)`

– *Description:* Checks that `Term`, a value comming from a text area is empty (can have spaces, newlines and linefeeds).

form_default/3: PREDICATE

Usage: `form_default(Val,Default,NewVal)`

– *Description:* Useful when a form is only partially filled, or when the executable can be invoked either by a link or by a form, to set form defaults. If the value of `Val` is empty then `NewVal=Default`, else `NewVal=Val`.
– *The following properties should hold at call time:*

`Val` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Default` is currently a term which is not a free variable. (term_typing:nonvar/1)

`NewVal` is a free variable. (term_typing:var/1)

set_cookie/2: PREDICATE

`set_cookie(Name,Value)`

Sets a cookie of name `Name` and value `Value`. Must be invoked before outputting any data, including the `cgi_reply` html-term.

Usage:

– *The following properties should hold at call time:*

`Name` is an atom. (basic_props:atm/1)

`Value` is an atomic term (an atom or a number). (basic_props:constant/1)

get_cookies/1: PREDICATE

`get_cookies(Cookies)`

Unifies `Cookies` with a dictionary of *attribute=value* pairs of the active cookies for this URL. If the flag `raw_form_values` is `on`, *value*s are always atoms even if they could be interpreted as numbers.

Usage:

– *The following properties should hold at call time:*

`Cookies` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`Cookies` is a dictionary of values. It is a list of pairs *atom=constant*. (pillow_types:value_dict/1)

url_query/2: PREDICATE

`url_query(Dict,URLArgs)`

(Deprecated, see `url_query_values/2`) Translates a dictionary `Dict` of parameter values into a string `URLArgs` for appending to a URL pointing to a form handler.

Usage:

– *The following properties should hold at call time:*

`Dict` is a dictionary of values. It is a list of pairs *atom=constant*. (pillow_types:value_dict/1)

`URLArgs` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`URLArgs` is a string (a list of character codes). (basic_props:string/1)

url_query_amp/2: PREDICATE

`url_query_amp(Dict,URLArgs)`

Translates a dictionary `Dict` of parameter values into a string `URLArgs` for appending to a URL pointing to a form handler to be used in the href of a link (uses & instead of &).

Usage:

– *The following properties should hold at call time:*

`Dict` is a dictionary of values. It is a list of pairs *atom=constant*. (pillow_types:value_dict/1)

`URLArgs` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`URLArgs` is a string (a list of character codes). (basic_props:string/1)

url_query_values/2: PREDICATE

`url_query_values(Dict,URLArgs)`

`Dict` is a dictionary of parameter values and `URLArgs` is the URL-encoded string of those assignments, which may appear after an URL pointing to a CGI script preceded by a '?'. `Dict` is computed according to the `raw_form_values` flag. The use of this predicate is reversible.

Usage 1:

– *The following properties should hold at call time:*

`Dict` is a dictionary of values. It is a list of pairs *atom=constant*. (pillow_types:value_dict/1)

`URLArgs` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`URLArgs` is a string (a list of character codes). (basic_props:string/1)

Usage 2:

– *The following properties should hold at call time:*

`Dict` is a free variable. (term_typing:var/1)

`URLArgs` is a string (a list of character codes). (basic_props:string/1)

– *The following properties hold upon exit:*

`Dict` is a dictionary of values. It is a list of pairs *atom=constant*. (pillow_types:value_dict/1)

my_url/1: PREDICATE

`my_url(URL)`

Unifies `URL` with the Uniform Resource Locator (WWW address) of this cgi executable.

Usage:

– *Calls should, and exit will be compatible with:*

`URL` is a string (a list of character codes). (basic_props:string/1)

– *The following properties hold upon exit:*

`URL` is a string (a list of character codes). (basic_props:string/1)

url_info/2: PREDICATE

`url_info(URL,URLTerm)`

Translates a URL `URL` to a Prolog structure `URLTerm` which details its various components, and vice-versa. For now non-HTTP URLs make the predicate fail.

Usage 1:

- *Calls should, and exit will be compatible with:*
  `URLTerm` specifies a URL. (pillow_types:url_term/1)
- *The following properties should hold at call time:*
  `URL` is an atom. (basic_props:atm/1)
- *The following properties hold upon exit:*
  `URLTerm` specifies a URL. (pillow_types:url_term/1)

Usage 2:

- *Calls should, and exit will be compatible with:*
  `URLTerm` specifies a URL. (pillow_types:url_term/1)
- *The following properties should hold at call time:*
  `URL` is a string (a list of character codes). (basic_props:string/1)
- *The following properties hold upon exit:*
  `URLTerm` specifies a URL. (pillow_types:url_term/1)

Usage 3:

- *The following properties should hold at call time:*
  `URL` is a free variable. (term_typing:var/1)
  `URLTerm` specifies a URL. (pillow_types:url_term/1)
- *The following properties hold upon exit:*
  `URL` is a string (a list of character codes). (basic_props:string/1)

url_info_relative/3: PREDICATE

`url_info_relative(URL,BaseURLTerm,URLTerm)`

Translates a relative URL `URL` which appears in the HTML page refered to by `BaseURLTerm` into `URLTerm`, a Prolog structure containing its absolute parameters. Absolute URLs are translated as with `url_info/2`. E.g.

```
url_info_relative("dadu.html",
                  http('www.foo.com',80,"/bar/scoob.html"), Info)
```

gives `Info = http('www.foo.com',80,"/bar/dadu.html")`.

Usage 1:

- *Calls should, and exit will be compatible with:*
  `URLTerm` specifies a URL. (pillow_types:url_term/1)
- *The following properties should hold at call time:*
  `URL` is an atom. (basic_props:atm/1)
  `BaseURLTerm` specifies a URL. (pillow_types:url_term/1)
- *The following properties hold upon exit:*
  `URLTerm` specifies a URL. (pillow_types:url_term/1)

Usage 2:

– *Calls should, and exit will be compatible with:*
`URLTerm` specifies a URL. (pillow_types:url_term/1)
– *The following properties should hold at call time:*
`URL` is a string (a list of character codes). (basic_props:string/1)
`BaseURLTerm` specifies a URL. (pillow_types:url_term/1)
– *The following properties hold upon exit:*
`URLTerm` specifies a URL. (pillow_types:url_term/1)

form_request_method/1: PREDICATE

Usage: `form_request_method(Method)`

– *Description:* Unifies `Method` with the method of invocation of the form handler (`GET` or `POST`).
– *The following properties hold upon exit:*
`Method` is an atom. (basic_props:atm/1)

icon_address/2: PREDICATE

`icon_address(Img,IAddress)`

The PiLLoW image `Img` has URL `IAddress`.

Usage:

– *Calls should, and exit will be compatible with:*
`Img` is an atom. (basic_props:atm/1)
`IAddress` is an atom. (basic_props:atm/1)
– *The following properties hold upon exit:*
`Img` is an atom. (basic_props:atm/1)
`IAddress` is an atom. (basic_props:atm/1)

html_protect/1: PREDICATE

`html_protect(Goal)`

Calls `Goal`. If an error occurs during its execution, or it fails, an HTML page is output informing about the incident. Normaly the whole execution of a CGI is protected thus.

Meta-predicate with arguments: `html_protect(goal)`.

Usage:

– *The following properties should hold at call time:*
`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

http_lines/3: PREDICATE

Usage: `http_lines(Lines,String,Tail)`

– *Description:* `Lines` is a list of the lines with occur in `String` until `Tail`. The lines may end UNIX-style or DOS-style in `String`, in `Lines` they have not end of line characters. Suitable to be used in DCGs.

– *Calls should, and exit will be compatible with:*
`Lines` is a list of `strings`. (basic_props:list/2)
`String` is a string (a list of character codes). (basic_props:string/1)
`Tail` is a string (a list of character codes). (basic_props:string/1)

132.3 Documentation on multifiles (`html`)

define_flag/3: PREDICATE

Defines a flag as follows:

```
define_flag(raw_form_values,[on,off],off).
```

(See Chapter 24 [Changing system behaviour and various flags], page 181).

If flag is `on`, values returned by `get_form_input/1` are always atoms, unchanged from its original value.

The predicate is *multifile*.

Usage: `define_flag(Flag,FlagValues,Default)`

– *The following properties hold upon exit:*
`Flag` is an atom. (basic_props:atm/1)
Define the valid flag values (basic_props:flag_values/1)

html_expansion/2: PREDICATE

The predicate is *multifile*.

Usage: `html_expansion(Term,Expansion)`

– *Description:* Hook predicate to define macros. Expand occurrences of `Term` into `Expansion`, in `output_html/1`. Take care to not transform something into itself!

132.4 Other information (`html`)

The code uses input from from L. Naish's forms and Francisco Bueno's previous Chat interface. Other people who have contributed are (please inform us if we leave out anybody): Markus Fromherz, Samir Genaim.

133 HTTP conectivity

Author(s): Daniel Cabeza.

This module implements the HTTP protocol, which allows retrieving data from HTTP servers.

133.1 Usage and interface (`http`)

- **Library usage:**
  `:- use_module(library(http)).`
- **Exports:**
  - *Predicates:*
    `fetch_url/3`.
- **Other modules used:**
  - *System library modules:*
    `strings`, `lists`, `pillow/pillow_aux`, `pillow/pillow_types`, `pillow/http_ll`.

133.2 Documentation on exports (`http`)

fetch_url/3: PREDICATE

`fetch_url(URL,Request,Response)`

Fetches the document pointed to by `URL` from Internet, using request parameters `Request`, and unifies `Response` with the parameters of the response. Fails on timeout. Note that redirections are not handled automatically, that is, if `Response` contains terms of the form `status(redirection,301,_)` and `location(NewURL)`, the program should in most cases access location `NewURL`.

Usage: `fetch_url(URL,Request,Response)`

- *The following properties should hold at call time:*
  `URL` specifies a URL. (pillow_types:url_term/1)
  `Request` is a list of `http_request_param`s. (basic_props:list/2)
- *The following properties hold upon exit:*
  `Response` is a list of `http_response_param`s. (basic_props:list/2)

134 PiLLoW types

Author(s): Daniel Cabeza.

Here are defined the regular types used in the documentation of the predicates of the PiLLoW package.

134.1 Usage and interface (pillow_types)

- **Library usage:**
  `:- use_module(library(pillow_types)).`
- **Exports:**
  - *Regular Types:*
    `canonic_html_term/1`, `canonic_xml_term/1`, `html_term/1`, `form_dict/1`, `form_assignment/1`, `form_value/1`, `value_dict/1`, `url_term/1`, `http_request_param/1`, `http_response_param/1`, `http_date/1`, `weekday/1`, `month/1`, `hms_time/1`.

134.2 Documentation on exports (pillow_types)

canonic_html_term/1: REGTYPE

A term representing HTML code in canonical, structured way. It is a list of terms defined by the following predicate:

```
canonic_html_item(comment(S)) :-
        string(S).
canonic_html_item(declare(S)) :-
        string(S).
canonic_html_item(env(Tag,Atts,Terms)) :-
        atm(Tag),
        list(Atts,tag_attrib),
        canonic_html_term(Terms).
canonic_html_item(Tag$Atts) :-
        atm(Tag),
        list(Atts,tag_attrib).
canonic_html_item(S) :-
        string(S).
tag_attrib(Att) :-
        atm(Att).
tag_attrib(Att=Val) :-
        atm(Att),
        string(Val).
```

Each structure represents one HTML construction:

env(*tag*,*attribs*,*terms***)**
: An HTML environment, with name *tag*, list of attributes *attribs* and contents *terms*.

$(*tag*,*attribs***)**
: An HTML element of name *tag* and list of attributes *attribs*. `($)/2` is defined by the pillow package as an infix, binary operator.

comment(*string***)**
: An HTML comment (translates to/from `<!--`*string*`-->`).

declare(*string***)**
: An HTML declaration, they are used only in the header (translates to/from `<!`*string*`>`).

string
: Normal text is represented as a list of character codes.

For example, the term

```
env(a,[href="www.therainforestsite.com"],
      ["Visit ",img$[src="TRFS.gif"]])
```

is output to (or parsed from):

```
<a href="www.therainforestsite.com">Visit <img src="TRFS.gif"></a>
```

Usage: `canonic_html_term(HTMLTerm)`

– *Description:* `HTMLTerm` is a term representing HTML code in canonical form.

canonic_xml_term/1: REGTYPE

A term representing XML code in canonical, structured way. It is a list of terms defined by the following predicate (see `tag_attrib/1` definition in `canonic_html_term/1`):

```
canonic_xml_item(Term) :-
        canonic_html_item(Term).
canonic_xml_item(xmldecl(Atts)) :-
        list(Atts,tag_attrib).
canonic_xml_item(env(Tag,Atts,Terms)) :-
        atm(Tag),
        list(Atts,tag_attrib),
        canonic_xml_term(Terms).
canonic_xml_item(elem(Tag,Atts)) :-
        atm(Tag),
        list(Atts,tag_attrib).
```

In addition to the structures defined by `canonic_html_term/1` (the `($)/2` structure appears only in malformed XML code), the following structures can be used:

elem(*tag***,***atts***)**
: Specifies an XML empty element of name *tag* and list of attributes *atts*. For example, the term

```
elem(arc,[weigh="3",begin="n1",end="n2"])
```

is output to (or parsed from):

```
<arc weigh="3" begin="n1" end="n2"/>
```

xmldecl(*atts***)**
: Specifies an XML declaration with attributes *atts* (translates to/from `<?xml` *atts*`?>`)

Usage: `canonic_xml_term(XMLTerm)`

– *Description:* `XMLTerm` is a term representing XML code in canonical form.

html_term/1: REGTYPE

A term which represents HTML or XML code in a structured way. In addition to the structures defined by `canonic_html_term/1` or `canonic_xml_term/1`, the following structures can be used:

begin(*tag***,***atts***)**
: It translates to the start of an HTML environment of name *tag* and attributes *atts*. There exists also a **begin(***tag***)** structure. Useful, in conjunction with the next structure, when including in a document output generated by an existing piece of code (e.g. *tag* = `pre`). Its use is otherwise discouraged.

end(*tag***)** Translates to the end of an HTML environment of name *tag*.

start Used at the beginning of a document (translates to `<html>`).

end Used at the end of a document (translates to `</html>`).

`--` Produces a horizontal rule (translates to `<hr>`).

`\\` Produces a line break (translates to `<br>`).

$ Produces a paragraph break (translates to `<p>`).

image(*address***)**
: Used to include an image of address (URL) *address* (equivalent to `img$[src=`*address*`]`).

image(*address***,***atts***)**
: As above with the list of attributes *atts*.

ref(*address***,***text***)**
: Produces a hypertext link, *address* is the URL of the referenced resource, *text* is the text of the reference (equivalent to `a([href=`*address*`],`*text*`)`).

label(*name***,***text***)**
: Labels *text* as a target destination with label *name* (equivalent to `a([name=`*name*`],`*text*`)`).

heading(*n***,***text***)**
: Produces a heading of level *n* (between 1 and 6), *text* is the text to be used as heading. Useful when one wants a heading level relative to another heading (equivalent to `h`*n*`(`*text*`)`).

itemize(*items***)**
: Produces a list of bulleted items, *items* is a list of corresponding HTML terms (translates to a `<ul>` environment).

enumerate(*items***)**
: Produces a list of numbered items, *items* is a list of corresponding HTML terms (translates to a `<ol>` environment).

description(*defs***)**
: Produces a list of defined items, *defs* is a list whose elements are definitions, each of them being a Prolog sequence (composed by `','/2` operators). The last element of the sequence is the definition, the other (if any) are the defined terms (translates to a `<dl>` environment).

nice_itemize(*img***,***items***)**
: Produces a list of bulleted items, using the image *img* as bullet. The predicate `icon_address/2` provides a colored bullet.

preformatted(*text*)
: Used to include preformatted text, *text* is a list of HTML terms, each element of the list being a line of the resulting document (translates to a `<pre>` environment).

verbatim(*text*)
: Used to include text verbatim, special HTML characters (`<`,`>`,`&`,`"` and space) are translated into its quoted HTML equivalent.

prolog_term(*term*)
: Includes any prolog term *term*, represented in functional notation. Variables are output as `_`.

nl
: Used to include a newline in the HTML source (just to improve human readability).

entity(*name*)
: Includes the entity of name *name* (ISO-8859-1 special character).

start_form(*addr*,*atts*)
: Specifies the beginning of a form. *addr* is the address (URL) of the program that will handle the form, and *atts* other attributes of the form, as the method used to invoke it. If *atts* is not present (there is only one argument) the method defaults to POST.

start_form
: Specifies the beginning of a form without assigning address to the handler, so that the form handler will be the cgi-bin executable producing the form.

end_form
: Specifies the end of a form.

checkbox(*name*,*state*)
: Specifies an input of type `checkbox` with name *name*, *state* is `on` if the checkbox is initially checked.

radio(*name*,*value*,*selected*)
: Specifies an input of type `radio` with name *name* (several radio buttons which are interlocked must share their name), *value* is the the value returned by the button, if *selected*=*value* the button is initially checked.

input(*type*,*atts*)
: Specifies an input of type *type* with a list of attributes *atts*. Possible values of *type* are `text`, `hidden`, `submit`, `reset`, ldots

textinput(*name*,*atts*,*text*)
: Specifies an input text area of name *name*. *text* provides the default text to be shown in the area, *atts* a list of attributes.

option(*name*,*val*,*options*)
: Specifies a simple option selector of name *name*, *options* is the list of available options and *val* is the initial selected option (if *val* is not in *options* the first item is selected by default) (translates to a `<select>` environment).

menu(*name*,*atts*,*items*)
: Specifies a menu of name *name*, list of attributes *atts* and list of options *items*. The elements of the list *items* are marked with the prefix operator `$` to indicate that they are selected (translates to a `<select>` environment).

form_reply

cgi_reply
: This two are equivalent, they do not generate HTML, rather, the CGI protocol requires this content descriptor to be used at the beginning by CGI executables (including form handlers) when replying (translates to `Content-type: text/html`).

pr Includes in the page a graphical logo with the message "Developed using the PiLLoW Web programming library", which points to the manual and library source.

name(*text*)
A term with functor *name*/1, different from the special functors defined herein, represents an HTML environment of name *name* and included text *text*. For example, the term

```
address('clip@clip.dia.fi.upm.es')
```

is translated into the HTML source

```
<address>clip@clip.dia.fi.upm.es</address>
```

name(*atts*,*text*)
A term with functor *name*/2, different from the special functors defined herein, represents an HTML environment of name *name*, attributes *atts* and included text *text*. For example, the term

```
a([href='http://www.clip.dia.fi.upm.es/'],"Clip home")
```

represents the HTML source

```
<a href="http://www.clip.dia.fi.upm.es/">Clip home</a>
```

Usage: `html_term(HTMLTerm)`
– *Description:* `HTMLTerm` is a term representing HTML code.

form_dict/1: REGTYPE

Usage: `form_dict(Dict)`
– *Description:* `Dict` is a dictionary of values of the attributes of a form. It is a list of `form_assignment`

form_assignment/1: REGTYPE

Usage: `form_assignment(Eq)`
– *Description:* `Eq` is an assignment of value of an attribute of a form. It is defined by:

```
form_assignment(A=V) :-
        atm(A),
        form_value(V).
form_value(A) :-
        atm(A).
form_value(N) :-
        num(N).
form_value(L) :-
        list(L,string).
```

form_value/1: REGTYPE

Usage: `form_value(V)`
– *Description:* `V` is a value of an attribute of a form.

value_dict/1: REGTYPE

Usage: `value_dict(Dict)`

– *Description:* `Dict` is a dictionary of values. It is a list of pairs *atom=constant*.

url_term/1: REGTYPE

A term specifying an Internet Uniform Resource Locator. Currently only HTTP URLs are supported. Example: `http('www.clip.dia.fi.upm.es',80,"/Software/Ciao/")`. Defined as

```
url_term(http(Host,Port,Document)) :-
        atm(Host),
        int(Port),
        string(Document).
```

Usage: `url_term(URL)`

– *Description:* `URL` specifies a URL.

http_request_param/1: REGTYPE

A parameter of an HTTP request:

- **head:** Specify that the document content is not wanted.
- **timeout(***T***):** *T* specifies the time in seconds to wait for the response. Default is 300 seconds.
- **if_modified_since(***Date***):** Get document only if newer than *Date*. *Date* has the format defined by `http_date/1`.
- **user_agent(***Agent***):** Provides a user-agent field, *Agent* is an atom. The string `"PiLLoW/1.1"` (or whatever version of PiLLoW is used) is appended.
- **authorization(***Scheme,Params***):** To provide credentials. See RFC 1945 for details.
- *option***(***Value***):** Any unary term, being *Value* an atom, can be used to provide another valid option (e.g. `from('user@machine')`).

Usage: `http_request_param(Request)`

– *Description:* `Request` is a parameter of an HTTP request.

http_response_param/1: REGTYPE

A parameter of an HTTP response:

- **content(***String***):** *String* is the document content (list of bytes). If the `head` parameter of the HTTP request is used, an empty list is get here.
- **status(***Type,Code,Reason***):** *Type* is an atom denoting the response type, *Code* is the status code (an integer), and *Reason* is a string holding the reason phrase.
- **message_date(***Date***):** *Date* is the date of the response, with format defined by `http_date/1`.
- **location(***Loc***):** This parameter appears when the document has moved, *Loc* is an atom holding the new location.
- **http_server(***Server***):** *Server* is the server responding, as a string.
- **authenticate(***Params***):** Returned if document is protected, *Params* is a list of chagenges. See RFC 1945 for details.
- **allow(***Methods***):** *Methods* are the methods allowed by the server, as a list of atoms.

- **content_encoding(***Encoding***):** *Encoding* is an atom defining the encoding.
- **content_length(***Length***):** *Length* is the length of the document (an integer).
- **content_type(***Type,Subtype,Params***):** Specifies the document content type, *Type* and *Subtype* are atoms, *Params* a list of parameters (e.g. `content_type(text,html,[])`).
- **expires(***Date***):** *Date* is the date after which the entity should be considered stale. Format defined by `http_date/1`.
- **last_modified(***Date***):** *Date* is the date at which the sender believes the resource was last modified. Format defined by `http_date/1`.
- **pragma(***String***):** Miscellaneous data.
- *header*(*String*)**:** Any other functor *header*/1 is an extension header.

Usage: `http_response_param(Response)`

– *Description:* `Response` is a parameter of an HTTP response.

http_date/1: REGTYPE

`http_date(Date)`

`Date` is a term defined as

```
http_date(date(WeekDay,Day,Month,Year,Time)) :-
        weekday(WeekDay),
        int(Day),
        month(Month),
        int(Year),
        hms_time(Time).
```

.

Usage: `http_date(Date)`

– *Description:* `Date` is a term denoting a date.

weekday/1: REGTYPE

Usage: `weekday(WeekDay)`

– *Description:* `WeekDay` is a term denoting a weekday.

month/1: REGTYPE

Usage: `month(Month)`

– *Description:* `Month` is a term denoting a month.

hms_time/1: REGTYPE

Usage: `hms_time(Time)`

– *Description:* `Time` is an atom of the form `hh:mm:ss`

135 Persistent predicate database

Author(s): José Manuel Gómez Pérez, Daniel Cabeza, Manuel Hermenegildo, The CLIP Group.

135.1 Introduction to persistent predicates

This library implements a *generic persistent predicate database*. The basic notion implemented by the library is that of a persistent predicate. The persistent predicate concept provides a simple, yet powerful generic persistent data access method [CHGT98,Par97]. A persistent predicate is a special kind of dynamic, data predicate that "resides" in some persistent medium (such as a set of files, a database, etc.) that is typically external to the program using such predicates. The main effect is that any changes made to a persistent predicate from a program "survive" across executions. I.e., if the program is halted and restarted the predicate that the new process sees is in precisely the same state as it was when the old process was halted (provided no change was made in the meantime to the storage by other processes or the user).

Persistent predicates appear to a program as ordinary predicates, and calls to these predicates can appear in clause bodies in the usual way. However, the definitions of these predicates do not appear in the program. Instead, the library maintains automatically the definitions of predicates which have been declared as persistent in the persistent storage.

Updates to persistent predicates can be made using enhanced versions of `asserta_fact/1`, `assertz_fact/1` and `retract_fact/1`. The library makes sure that each update is a transactional update, in the sense that if the update terminates, then the permanent storage has definitely been modified. For example, if the program making the updates is halted just after the update and then restarted, then the updated state of the predicate will be seen. This provides security against possible data loss due to, for example, a system crash. Also, due to the atomicity of the transactions, persistent predicates allow concurrent updates from several programs.

135.2 Persistent predicates, files, and relational databases

The concept of persistent predicates provided by this library essentially implements a lightweight, simple, and at the same time powerful form of relational database (a deductive database), and which is standalone, in the sense that it does not require external support, other than the file management capabilities provided by the operating system. This is due to the fact that the persistent predicates are in fact stored in one or more auxiliary files below a given directory.

This type of database is specially useful when building small to medium-sized standalone applications in Prolog which require persistent storage. In many cases it provides a much easier way of implementing such storage than using files under direct program control. For example, interactive applications can use persistent predicates to represent their internal state in a way that is close to the application. The persistence of such predicates then allows automatically restoring the state to that at the end of a previous session. Using persistent predicates amounts to simply declaring some predicates as such and eliminates having to worry about opening files, closing them, recovering from system crashes, etc.

In other cases, however, it may be convenient to use a relational database as persistent storage. This may be the case, for example, when the data already resides in such a database (where it is perhaps accessed also by other applications) or the volume of data is very large. `persdb_sql` [CCG98] is a companion library which implements the same notion of persistent predicates used herein, but keeping the storage in a relational database. This provides a very natural and transparent way to access SQL database relations from a Prolog program. In that library, facilities are also provided for reflecting more complex *views* of the database relations

as predicates. Such views can be constructed as conjunctions, disjunctions, projections, etc. of database relations, and may include SQL-like aggregation operations.

A nice characteristic of the notion of persistent predicates used in both of these libraries is that it abstracts away how the predicate is actually stored. Thus, a program can use persistent predicates stored in files or in external relational databases interchangeably, and the type of storage used for a given predicate can be changed without having to modify the program (except for replacing the corresponding `persistent/2` declarations).

An example application of the `persdb` and `persdb_sql` libraries (and also the `pillow` library [CH97]), is `WebDB` [GCH98]. `WebDB` is a generic, highly customizable *deductive database engine* with an *html interface*. `WebDB` allows creating and maintaining Prolog-based databases as well as relational databases (residing in conventional relational database engines) using any standard WWW browser.

135.3 Using file-based persistent predicates

Persistent predicates can be declared statically, using `persistent/2` declarations (which is the preferred method, when possible), or dynamically via calls to `make_persistent/2`. Currently, persistent predicates may only contain facts, i.e., they are *dynamic* predicates of type `data/1`.

Predicates declared as persistent are linked to directory, and the persistent state of the predicate will be kept in several files below that directory. The files in which the persistent predicates are stored are in readable, plain ASCII format, and in Prolog syntax. One advantage of this approach is that such files can also be created or edited by hand, in a text editor, or even by other applications.

An example definition of a persistent predicate implemented by files follows:

```
:- persistent(p/3,dbdir).

persistent_dir(dbdir, '/home/clip/public_html/db').
```

The first line declares the predicate `p/3` persistent. The argument `dbdir` is a key used to index into a fact of the relation `persistent_dir/2-4`, which specifies the directory where the corresponding files will be kept. The effect of the declaration, together with the `persistent_dir/2-4` fact, is that, although the predicate is handled in the same way as a normal data predicate, in addition the system will create and maintain efficiently a persistent version of `p/3` via files in the directory `/home/clip/public_html/db`.

The level of indirection provided by the `dbdir` argument makes it easy to place the storage of several persistent predicates in a common directory, by specifying the same key for all of them. It also allows changing the directory for several such persistent predicates by modifying only one fact in the program. Furthermore, the `persistent_dir/2-4` predicate can even be dynamic and specified at run-time.

135.4 Implementation Issues

We outline the current implementation approach. This implementation attempts to provide at the same time efficiency and security. To this end, up to three files are used for each predicate (the persistence set): the data file, the operations file, and the backup file. In the updated state the facts (tuples) that define the predicate are stored in the data file and the operations file is empty (the backup file, which contains a security copy of the data file, may or may not exist).

While a program using a persistent predicate is running, any insertion (assert) or deletion (retract) operations on the predicate are performed on both the program memory and on the persistence set. However, in order to incurr only a small overhead in the execution, rather than changing the data file directly, a record of each of the insertion and deletion operations is

appended to the operations file. The predicate is then in a transient state, in that the contents of the data file do not reflect exactly the current state of the corresponding predicate. However, the complete persistence set does.

When a program starts, all pending operations in the operations file are performed on the data file. A backup of the data file is created first to prevent data loss if the system crashes during this operation. The order in which this updating of files is done ensures that, if at any point the process dies, on restart the data will be completely recovered. This process of updating the persistence set can also be triggered at any point in the execution of the program (for example, when halting) by calling `update_files`.

135.5 Defining an initial database

It is possible to define an initial database by simply including in the program code facts of persistent predicates. They will be included in the persistent database when it is created. They are ignored in successive executions.

135.6 Using persistent predicates from the top level

Special care must be taken when loading into the top level modules or user files which use persistent predicates. Beforehand, a goal `use_module(library(persdb(persdbrt)))` must be issued. Furthermore, since persistent predicates defined by the loaded files are in this way defined dynamically, a call to `initialize_db/0` is commonly needed after loading and before calling predicates of these files.

135.7 Usage and interface (`persdbrt`)

- **Library usage:**
  There are two packages which implement persistence: `persdb` and `'persdb/ll'` (for low level). In the first, the standard builtins `asserta_fact/1`, `assertz_fact/1`, and `retract_fact/1` are replaced by new versions which handle persistent data predicates, behaving as usual for normal data predicates. In the second package, predicates with names starting with `p` are defined, so that there is no overhead in calling the standard builtins. In any case, each package is used as usual: including it in the package list of the module, or using the `use_package/1` declaration.
- **Exports:**
  - *Predicates:*
    `passerta_fact/1`, `passertz_fact/1`, `pretract_fact/1`, `pretractall_fact/1`, `asserta_fact/1`, `assertz_fact/1`, `retract_fact/1`, `retractall_fact/1`, `initialize_db/0`, `make_persistent/2`, `update_files/0`, `update_files/1`, `create/2`.
  - *Regular Types:*
    `meta_predname/1`, `directoryname/1`.
  - *Multifiles:*
    `$is_persistent/2`, `persistent_dir/2`, `persistent_dir/4`.
- **Other modules used:**
  - *System library modules:*
    `lists`, `read`, `aggregates`, `system`, `file_locks/file_locks`, `persdb/persdbcache`.

135.8 Documentation on exports (persdbrt)

passerta_fact/1: PREDICATE

Meta-predicate with arguments: `passerta_fact(fact)`.

Usage: `passerta_fact(Fact)`

- *Description:* Persistent version of `asserta_fact/1`: the current instance of `Fact` is interpreted as a fact (i.e., a relation tuple) and is added at the beginning of the definition of the corresponding predicate. The predicate concerned must be declared `persistent`. Any uninstantiated variables in the `Fact` will be replaced by new, private variables. Defined in the `'persdb/ll'` package.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

passertz_fact/1: PREDICATE

Meta-predicate with arguments: `passertz_fact(fact)`.

Usage: `passertz_fact(Fact)`

- *Description:* Persistent version of `assertz_fact/1`: the current instance of `Fact` is interpreted as a fact (i.e., a relation tuple) and is added at the end of the definition of the corresponding predicate. The predicate concerned must be declared `persistent`. Any uninstantiated variables in the `Fact` will be replaced by new, private variables. Defined in the `'persdb/ll'` package.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

pretract_fact/1: PREDICATE

`pretract_fact(P)`

Retracts a predicate in both, the dynamic and the persistent databases.

Meta-predicate with arguments: `pretract_fact(fact)`.

Usage: `pretract_fact(Fact)`

- *Description:* Persistent version of `retract_fact/1`: deletes on backtracking all the facts which unify with `Fact`. The predicate concerned must be declared `persistent`. Defined in the `'persdb/ll'` package.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

pretractall_fact/1: PREDICATE

`pretractall_fact(P)`

Retracts all the instances of a predicate in both, the dynamic and the persistent databases.

Meta-predicate with arguments: `pretractall_fact(fact)`.

asserta_fact/1: PREDICATE

Meta-predicate with arguments: `asserta_fact(fact)`.

Usage: `asserta_fact(Fact)`

- *Description:* Same as `passerta_fact/1`, but if the predicate concerned is not persistent then behaves as the builtin of the same name. Defined in the `persdb` package.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

assertz_fact/1: PREDICATE

Meta-predicate with arguments: `assertz_fact(fact)`.

Usage: `assertz_fact(Fact)`

- *Description:* Same as `passertz_fact/1`, but if the predicate concerned is not persistent then behaves as the builtin of the same name. Defined in the `persdb` package.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

retract_fact/1: PREDICATE

Meta-predicate with arguments: `retract_fact(fact)`.

Usage: `retract_fact(Fact)`

- *Description:* Same as `pretract_fact/1`, but if the predicate concerned is not persistent then behaves as the builtin of the same name. Defined in the `persdb` package.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

retractall_fact/1: PREDICATE

Meta-predicate with arguments: `retractall_fact(fact)`.

Usage: `retractall_fact(Fact)`

- *Description:* Same as `pretractall_fact/1`, but if the predicate concerned is not persistent then behaves as the builtin of the same name. Defined in the `persdb` package.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

initialize_db/0: PREDICATE

Usage:

- *Description:* Initializes the whole database, updating the state of the declared persistent predicates. Must be called explicitly after dynamically defining clauses for `persistent_dir/2`.

make_persistent/2: PREDICATE

Meta-predicate with arguments: `make_persistent(spec,?)`.

Usage: `make_persistent(PredDesc,Keyword)`

- *Description:* Dynamic version of the `persistent` declaration.
- *The following properties should hold at call time:*

  persdbrt:meta_predname(PredDesc) (persdbrt:meta_predname/1)

  `Keyword` is an atom corresponding to a directory identifier. (persdbcache:keyword/1)

update_files/0: PREDICATE

Usage:

- *Description:* Updates the files comprising the persistence set of all persistent predicates defined in the application.

update_files/1: PREDICATE

Meta-predicate with arguments: `update_files(list(spec))`.

Usage: `update_files(PredSpecList)`

- *Description:* Updates the files comprising the persistence set of the persistent predicates in `PredSpecList`.
- *Call and exit should be compatible with:*

  `PredSpecList` is a list of `prednames`. (basic_props:list/2)

create/2: PREDICATE

No further documentation available for this predicate.

meta_predname/1: REGTYPE

A regular type, defined as follows:

```
meta_predname($:(P)) :-
        predname(P).
```

directoryname/1: REGTYPE

Usage: `directoryname(X)`

- *Description:* `X` is an atom, the name of a directory.

135.9 Documentation on multifiles (persdbrt)

$is_persistent/2: PREDICATE

`$is_persistent(Spec,Key)`

Predicate `Spec` persists within database `Key`. Programmers should not define this predicate directly in the program.

The predicate is *multifile*.

The predicate is of type *data*.

persistent_dir/2: PREDICATE

The predicate is *multifile*.

The predicate is of type *data*.

Usage: `persistent_dir(Keyword,Location_Path)`

- *Description:* Relates identifiers of locations (the `Keyword`s) with descriptions of such locations (`Location_Path`s). `Location_Path` is **a directory** and it means that the definition for the persistent predicates associated with `Keyword` is kept in files below that directory (which must previously exist). These files, in the updated state, contain the actual definition of the predicate in Prolog syntax (but with module names resolved).
- *Call and exit should be compatible with:*

  `Keyword` is an atom corresponding to a directory identifier. (persdbcache:keyword/1)

  `Location_Path` is an atom, the name of a directory. (persdbrt:directoryname/1)

persistent_dir/4: PREDICATE

The predicate is *multifile*.

The predicate is of type *data*.

Usage: `persistent_dir(Keyword,Location_Path,DirPerms,FilePerms)`

- *Description:* The same as `persistent_dir/2`, but including also the permission modes for persistent directories and files.
- *Call and exit should be compatible with:*

  `Keyword` is an atom corresponding to a directory identifier. (persdbcache:keyword/1)

  `Location_Path` is an atom, the name of a directory. (persdbrt:directoryname/1)

  `DirPerms` is an integer. (basic_props:int/1)

  `FilePerms` is an integer. (basic_props:int/1)

135.10 Documentation on internals (persdbrt)

persistent/2: DECLARATION

Usage: `:- persistent(PredDesc,Keyword).`

- *Description:* Declares the predicate `PredDesc` as persistent. `Keyword` is the identifier of a location where the persistent storage for the predicate is kept. The location `Keyword` is described in the `persistent_dir` predicate, which must contain a fact in which the first argument unifies with `Keyword`.

– *The following properties should hold upon exit:*
`PredDesc` is a Name/Arity structure denoting a predicate name:

```
predname(P/A) :-
        atm(P),
        nnegint(A).
```

(basic_props:predname/1)
`Keyword` is an atom corresponding to a directory identifier. (persdbcache:keyword/1)

keyword/1: PREDICATE

An atom which identifies a fact of the `persistent_dir/2` relation. This fact relates this atom to a directory in which the persistent storage for one or more persistent predicates is kept. Storage is expected under a subdirectory by the name of the module and in a file by the name of the predicate.

135.11 Known bugs and planned improvements (`persdbrt`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.
- To load in the toplevel a file which uses this package, module `library(persdb(persdbrt))` has to be previously loaded.

136 Using the persdb library

Author(s): The CLIP Group.

Through the following examples we will try to illustrate the two mains ways of declaring and using persistent predicates: statically (the preferred method) and dynamically (necessary when the new persistent predicates have to be defined at run-time). The final example is a small application implementing a simple persistent queue.

136.1 An example of persistent predicates (static version)

136.2 An example of persistent predicates (dynamic version)

136.3 A simple application / a persistent queue

137 Filed predicates

Author(s): Francisco Bueno.

This package allows using files as a " cache" for predicates defined by facts. This is useful for huge tables of facts that may push the memory limits of the system too far. Goals of a filed predicate are executed simply by reading from the corresponding file.

Anything in the DB file used for the predicate that is different from a fact for the corresponding predicate is ignored. Each call to a filed predicate forces opening the file, so the use of this package is subject to the limit on the number of open files that the system can support.

Dynamic modification of the filed predicates is also allowed during execution of the program. Thus filed predicates are regarded as dynamic, data predicates residing in a file. However, dynamic modifications to the predicates do not affect the file, unless the predicate is also declared persistent.

The package is compatible with `persdb` in the sense that a predicate can be made both filed and persistent. In this way, the predicate can be used in programs, but it will not be loaded (saving memory), can also be modified during execution, and modifications will persist in the file. Thus, the user interface to both packages is the same (so the DB file must be one for both filing and persistency).

137.1 Usage and interface (`factsdb_doc`)

- **Library usage:**
  This facility is used as a package, thus either including `factsdb` in the package list of the module, or by using the `use_package/1` declaration. The facility predicates are defined in library module `factsdb_rt`.
- **Other modules used:**
  - *System library modules:*
    `factsdb/factsdb_rt`.

137.2 Documentation on multifiles (`factsdb_doc`)

$factsdb$cached_goal/3: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

137.3 Known bugs and planned improvements (`factsdb_doc`)

- The DB files for persistent predicates have to be used as such from the beginning. Using a DB file for a filed predicate first, and then using it also when making the predicate persistent won't work. Nor the other way around: using a DB file for a persistent predicate first, and then using it also when making the predicate filed.

138 Filed predicates (runtime)

Author(s): Francisco Bueno.

Runtime module for the `factsdb` package.

138.1 Usage and interface (`factsdb_rt`)

- **Library usage:**
  `:- use_module(library(factsdb_rt)).`
- **Exports:**
  - *Predicates:*
    `asserta_fact/1`, `assertz_fact/1`, `call/1`, `current_fact/1`, `retract_fact/1`.
  - *Multifiles:*
    `$factsdb$cached_goal/3`, `persistent_dir/2`, `file_alias/2`.
- **Other modules used:**
  - *System library modules:*
    `counters`, `read`, `persdb/persdbcache`.

138.2 Documentation on exports (`factsdb_rt`)

asserta_fact/1: PREDICATE

Meta-predicate with arguments: `asserta_fact(fact)`.

Usage: `asserta_fact(Fact)`

- *Description:* Version of `data_facts:asserta_fact/1` for filed predicates. The current instance of `Fact` is interpreted as a fact and is added at the beginning of the definition of the corresponding predicate. Therefore, before all the facts filed in the DB file for the predicate. The predicate concerned must be declared as `facts`; if it is not, then `data_facts:asserta_fact/1` is used.
- *The following properties should hold at call time:*
  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

assertz_fact/1: PREDICATE

Meta-predicate with arguments: `assertz_fact(fact)`.

Usage: `assertz_fact(Fact)`

- *Description:* Version of `data_facts:assertz_fact/1` for filed predicates. The current instance of `Fact` is interpreted as a fact and is added at the end of the definition of the corresponding predicate. Therefore, after all the facts filed in the DB file for the predicate. The predicate concerned must be declared as `facts`; if it is not, then `data_facts:assertz_fact/1` is used.
- *The following properties should hold at call time:*
  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

call/1: PREDICATE

Meta-predicate with arguments: `call(fact)`.

Usage: `call(Fact)`

- *Description:* Same as `current_fact/1` if the predicate concerned is declared as `facts`. If it is not, an exception is raised.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

current_fact/1: PREDICATE

Meta-predicate with arguments: `current_fact(fact)`.

Usage: `current_fact(Fact)`

- *Description:* Version of `data_facts:current_fact/1` for filed predicates. The current instance of `Fact` is interpreted as a fact and is unified with an actual fact in the current definition of the corresponding predicate. Therefore, with a fact previously asserted or filed in the DB file for the predicate, if it has not been retracted. The predicate concerned must be declared as `facts`; if it is not, then `data_facts:current_fact/1` is used.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

retract_fact/1: PREDICATE

Meta-predicate with arguments: `retract_fact(fact)`.

Usage: `retract_fact(Fact)`

- *Description:* Version of `data_facts:retract_fact/1` for filed predicates. The current instance of `Fact` is interpreted as a fact and is unified with an actual fact in the current definition of the corresponding predicate; such a fact is deleted from the predicate definition. This is true even for the facts filed in the DB file for the predicate; but these are NOT deleted from the file (unless the predicate is persistent). The predicate concerned must be declared as `facts`; if it is not, then `data_facts:retract_fact/1` is used.
- *The following properties should hold at call time:*

  `Fact` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

138.3 Documentation on multifiles (`factsdb_rt`)

$factsdb$cached_goal/3: PREDICATE

`$factsdb$cached_goal(Spec,Spec,Key)`

Predicate `Spec` is filed within database `Key`. Programmers should not define this predicate directly in the program.

The predicate is *multifile*.

persistent_dir/2: PREDICATE

See `persdb`.

The predicate is *multifile*.

The predicate is of type *data*.

file_alias/2: PREDICATE

See `symfnames`. This predicate is used only if `persistent_dir/2` fails.

The predicate is *multifile*.

The predicate is of type *data*.

138.4 Documentation on internals (`factsdb_rt`)

facts/2: DECLARATION

Usage: `:- facts(PredDesc,Keyword).`

- *Description:* Declares the predicate `PredDesc` as filed. `Keyword` is the identifier of a location where the file DB for the predicate is kept. The location `Keyword` is described in the `file_alias` predicate, which must contain a fact in which the first argument unifies with `Keyword`.
- *The following properties should hold upon exit:*

  `PredDesc` is a Name/Arity structure denoting a predicate name:

  ```
  predname(P/A) :-
          atm(P),
          nnegint(A).
  ```

  (basic_props:predname/1)

  `Keyword` is an atom corresponding to a directory identifier. (persdbcache:keyword/1)

keyword/1: PREDICATE

See `persdbrt`. The same conventions for location of DB files apply in both packages.

139 sqltypes (library)

139.1 Usage and interface (sqltypes)

- **Library usage:**
  `:- use_module(library(sqltypes)).`
- **Exports:**
  - *Predicates:*
    `accepted_type/2`, `get_type/2`, `type_compatible/2`, `type_union/3`, `sybase2sqltypes_list/2`, `sybase2sqltype/2`, `postgres2sqltypes_list/2`, `postgres2sqltype/2`.
  - *Regular Types:*
    `sqltype/1`, `sybasetype/1`, `postgrestype/1`.

139.2 Documentation on exports (sqltypes)

sqltype/1: REGTYPE

```
sqltype(int).
sqltype(flt).
sqltype(num).
sqltype(string).
sqltype(date).
sqltype(time).
sqltype(datetime).
```

These types have the same meaning as the corresponding standard types in the `basictypes` library.

Usage: `sqltype(Type)`

- *Description:* `Type` is an SQL data type supported by the translator.

accepted_type/2: PREDICATE

Usage: `accepted_type(SystemType,NativeType)`

- *Description:* For the moment, tests wether the `SystemType` received is a sybase or a postgres type (in the future other systems should be supported) and obtains its equivalent `NativeType` sqltype.
- *Call and exit should be compatible with:*
  `SystemType` is an SQL data type supported by Sybase. (sqltypes:sybasetype/1)
  `NativeType` is an SQL data type supported by the translator. (sqltypes:sqltype/1)

get_type/2: PREDICATE

Usage: `get_type(Constant,Type)`

- *Description:* Prolog implementation-specific definition of type retrievals. CIAO Prolog version given here (ISO).

– *Call and exit should be compatible with:*
`Constant` is any term. (basic_props:term/1)
`Type` is an SQL data type supported by the translator. (sqltypes:sqltype/1)
– *The following properties should hold at call time:*
`Constant` is currently a term which is not a free variable. (term_typing:nonvar/1)

type_compatible/2: PREDICATE

Usage: `type_compatible(TypeA,TypeB)`

– *Description:* Checks if `TypeA` and `TypeB` are compatible types, i.e., they are the same or one is a subtype of the other.
– *Call and exit should be compatible with:*
`TypeA` is an SQL data type supported by the translator. (sqltypes:sqltype/1)
`TypeB` is an SQL data type supported by the translator. (sqltypes:sqltype/1)

type_union/3: PREDICATE

Usage: `type_union(TypeA,TypeB,Union)`

– *Description:* `Union` is the union type of `TypeA` and `TypeB`.
– *Call and exit should be compatible with:*
`TypeA` is an SQL data type supported by the translator. (sqltypes:sqltype/1)
`TypeB` is an SQL data type supported by the translator. (sqltypes:sqltype/1)
`Union` is an SQL data type supported by the translator. (sqltypes:sqltype/1)

sybasetype/1: REGTYPE

SQL datatypes supported by Sybase for which a translation is defined:

```
sybasetype(integer).
sybasetype(numeric).
sybasetype(float).
sybasetype(double).
sybasetype(date).
sybasetype(char).
sybasetype(varchar).
sybasetype('long varchar').
sybasetype(binary).
sybasetype('long binary').
sybasetype(timestamp).
sybasetype(time).
sybasetype(tinyint).
```

Usage: `sybasetype(Type)`

– *Description:* `Type` is an SQL data type supported by Sybase.

sybase2sqltypes_list/2: PREDICATE

Usage: `sybase2sqltypes_list(SybaseTypesList,SQLTypesList)`

– *Description:* `SybaseTypesList` is a list of Sybase SQL types. `PrologTypesList` contains their equivalent SQL-type names in CIAO.
– *The following properties should hold upon exit:*
`SybaseTypesList` is a list. (basic_props:list/1)
`SQLTypesList` is a list. (basic_props:list/1)

sybase2sqltype/2: PREDICATE

Usage: `sybase2sqltype(SybaseType,SQLType)`

– *Description:* `SybaseType` is a Sybase SQL type name, and `SQLType` is its equivalent SQL-type name in CIAO.
– *The following properties should hold upon exit:*
`SybaseType` is an SQL data type supported by Sybase. (sqltypes:sybasetype/1)
`SQLType` is an SQL data type supported by the translator. (sqltypes:sqltype/1)

postgrestype/1: REGTYPE

SQL datatypes supported by postgreSQL for which a translation is defined:

```
postgrestype(int2).
postgrestype(int4).
postgrestype(int8).
postgrestype(float4).
postgrestype(float8).
postgrestype(date).
postgrestype(timestamp).
postgrestype(time).
postgrestype(char).
postgrestype(varchar).
postgrestype(text).
postgrestype(bool).
```

Usage: `postgrestype(Type)`

– *Description:* `Type` is an SQL data type supported by postgres.

postgres2sqltypes_list/2: PREDICATE

Usage: `postgres2sqltypes_list(PostgresTypesList,SQLTypesList)`

– *Description:* `PostgresTypesList` is a list of postgres SQL types. `PrologTypesList` contains their equivalent SQL-type names in CIAO.
– *The following properties should hold upon exit:*
`PostgresTypesList` is a list. (basic_props:list/1)
`SQLTypesList` is a list. (basic_props:list/1)

postgres2sqltype/2: PREDICATE

Usage: `postgres2sqltype(PostgresType,SQLType)`

– *Description:* `PostgresType` is a postgres SQL type name, and `SQLType` is its equivalent SQL-type name in CIAO.

– *The following properties should hold upon exit:*
`PostgresType` is an SQL data type supported by postgres. (sqltypes:postgrestype/1)
`SQLType` is an SQL data type supported by the translator. (sqltypes:sqltype/1)

140 persdbtr_sql (library)

140.1 Usage and interface (persdbtr_sql)

- **Library usage:**
  `:- use_module(library(persdbtr_sql)).`
- **Exports:**
  - *Predicates:*
    `sql_persistent_tr/2`, `sql_goal_tr/2`, `dbId/2`.

140.2 Documentation on exports (persdbtr_sql)

sql_persistent_tr/2: PREDICATE

No further documentation available for this predicate.

sql_goal_tr/2: PREDICATE

No further documentation available for this predicate.

dbId/2: PREDICATE

No further documentation available for this predicate.

The predicate is of type *data*.

141 pl2sqlinsert (library)

141.1 Usage and interface (pl2sqlinsert)

- **Library usage:**
  `:- use_module(library(pl2sqlinsert)).`
- **Exports:**
  - *Predicates:*
    `pl2sqlInsert/2`.
  - *Multifiles:*
    `sql__relation/3`, `sql__attribute/4`.
- **Other modules used:**
  - *System library modules:*
    `operators`, `default_predicates`, `lists`.

141.2 Documentation on exports (pl2sqlinsert)

pl2sqlInsert/2: PREDICATE

No further documentation available for this predicate.

141.3 Documentation on multifiles (pl2sqlinsert)

sql__relation/3: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

sql__attribute/4: PREDICATE

No further documentation available for this predicate.

The predicate is *multifile*.

The predicate is of type *data*.

142 Prolog/Java Bidirectional Interface

Author(s): Jesús Correas, The CLIP Group.

The increasing diversity of platforms used today and the diffusion of Internet and the World Wide Web makes compatibility between platforms a key factor to run the software everywhere with no change. Java seems to achieve this goal, using a bytecode intermediate language and a large library of platform-dependent and independent classes which fully implements many. On the other hand, Prolog provides a powerful implementation of logic programming paradigm. This document includes the reference manual of the Prolog/Java bidirectional interface implemented in Ciao. In addition, it has been developed an application of this interface that makes use of an object oriented extension of Prolog to encapsulate the java classes, O'Ciao, both the ones defined in the JDK as well as new classes developed in Java. These classes can be used in the object oriented prolog extension of Ciao just like native O'Ciao classes.

The proposed interaction between both languages is realized as an interface between two processes, a Java process and a Prolog process, running separately. This approach allows the programmer to use of both Java and Prolog, without the compiler-dependent glue code used in other linkage-oriented approaches, and preserves the philosophy of Java as an independent language. The interface communication is based on a clean socket-based protocol, providing hardware and software independence. This allows also both processes to be run in different machines connected by a TCP/IP transport protocol, based on a client/server model that can evolve to a more cooperative model.

The present manual includes reference information about the Prolog side of the bidirectional Java/Prolog interface. The Java side of this interface is explained in the HTML pages generated by Javadoc.

142.1 Distributed Programming Model

The differences between Prolog and Java impose the division of the interface in two main parts: a prolog-to-java and a java-to-prolog interfaces. Most of the applications that will use this interface will consider that will be a "client' side that request actions and queries to a "server' side, which accomplish the actions and answer the queries. In a first approach, any of the both one-way interfaces implement a pure client/server model: the server waits for a query, performs the received query and sleeps until the next query comes; the client starts the server, carries out the initial part of the job initiating all the conversations with the server, and requests the server to do some things sometimes.

This model cannot handle correctly the tasks regarding an event oriented programming environment like java. A usual application of the prolog-to-java interface could be a graphical user interface server made in java, and a prolog client on the other side. A pure client/server model based on requests and results is not powerful enough to leave the prolog side managing all the application specific work of this example: some java specific stuff is needed to catch and manipulate properly the events thrown by the graphical user interface. This problem can be solved in a distributed context, on which both languages are clients and servers simultaneously, and can perform requests and do actions at a time. Using this model, the prolog side can add a prolog goal as listener of a specific event, and the java side launches that goal when the event raises.

In any case, the client/server approach simplifies the design of the interface, so both interfaces have been designed in such way, but keeping in mind that the goal is to reach a distributed environment, so each side do the things it is best designed for.

143 Prolog to Java interface

Author(s): Jesús Correas.

This module defines the Ciao Prolog to Java interface. This interface allows a Prolog program to start a Java process, create Java objects, invoke methods, set/get attributes (fields), and handle Java events.

This interface only works with JDK version 1.2 or higher.

Although the Java side interface is explained in Javadoc format (it is available at `library/javall/javadoc/` in your Ciao installation), the general interface structure is detailed here.

143.1 Prolog to Java Interface Structure

This interface is made up of two parts: a Prolog part and a Java part, running in separate processes. The Prolog part receives requests from a Prolog program and sends them to the Java part through a socket. The Java part receives requests from the socket and performs the actions included in the requests.

If an event is thrown in the Java side, an asynchronous message must be sent away to the Prolog side, in order to launch a Prolog goal to handle the event. This asynchronous communication is performed using a separate socket. The nature of this communication needs the use of threads both in Java and Prolog: to deal with the 'sequential program flow,' and other threads for event handling.

In both sides the threads are automatically created by the context of the objects we use. The user must be aware that different requests to the other side of the interface could run concurrently.

143.1.1 Prolog side of the Java interface

The Prolog side receives the actions to do in the Java side from the user program, and sends them to the Java process through the socket connection. When the action is done in the Java side, the result is returned to the user Prolog program, or the action fails if there is any problem in the Java side.

Prolog data representation of Java elements is very simple in this interface. Java primitive types such as integers and characters are translated into the Prolog corresponding terms, and even some Java objects are translated in the same way (e. g. Java strings). Java objects are represented in Prolog as compound terms with a reference id to identify the corresponding Java object. Data conversion is made automatically when the interface is used, so the Prolog user programs do not have to deal with the complexity of these tasks.

143.1.2 Java side

The Java side of this layer is more complex than the Prolog side. The tasks this part has to deal with are the following:

- Wait for requests from the Prolog side.
- Translate the Prolog terms received in the Prolog 'serialized' form to a more useful Java representation (see the Java interface documentation available at `library/javall/javadoc/` in your Ciao installation for details regarding Java representation of Prolog terms).
- Interpret the requests received from the Prolog side, and execute them.
- Handle the set of objects created by or derived from the requests received from the prolog side.

- Handle the events raised in the Java side, and launch the listeners added in the prolog side.
- Handle the exceptions raised in the Java side, and send them to the Prolog side.

In the implementation of the Java side, two items must be carefully designed: the handling of Java objects, and the representation of prolog data structures. The last item is specially important because all the interactions between Prolog and Java are made using Prolog structures, an easy way to standardize the different data management in both sides. Even the requests themselves are encapsulated using Prolog structures. The overload of this encapsulation is not significant in terms of socket traffic, due to the optimal implementation of the prolog serialized term.

The java side must handle the objects created from the Prolog side dinamically, and these objects must be accessed as fast as possible from the set of objects. The Java API provides a powerful implementation of Hash tables that achieves all the requirements of our implementation.

On the other hand, the java representation of prolog terms is made using the inheritance of java classes. In the java side there exists a representation of a generic prolog term, implemented as an abstract class in java. Variables, atoms, compound terms, lists, and numeric terms are classes in the java side which inherit from the term class. Java objects can be seen also under the prolog representation as compound terms, where the single argument corresponds to the Hash key of the actual java object in the Hash table referred to before. This behaviour makes the handling of mixed java and prolog elements easy. Prolog goals are represented in the java side as objects which contain a prolog compound term with the term representing the goal. This case will be seen more in depth in next chapter, where the java to prolog interface is explained.

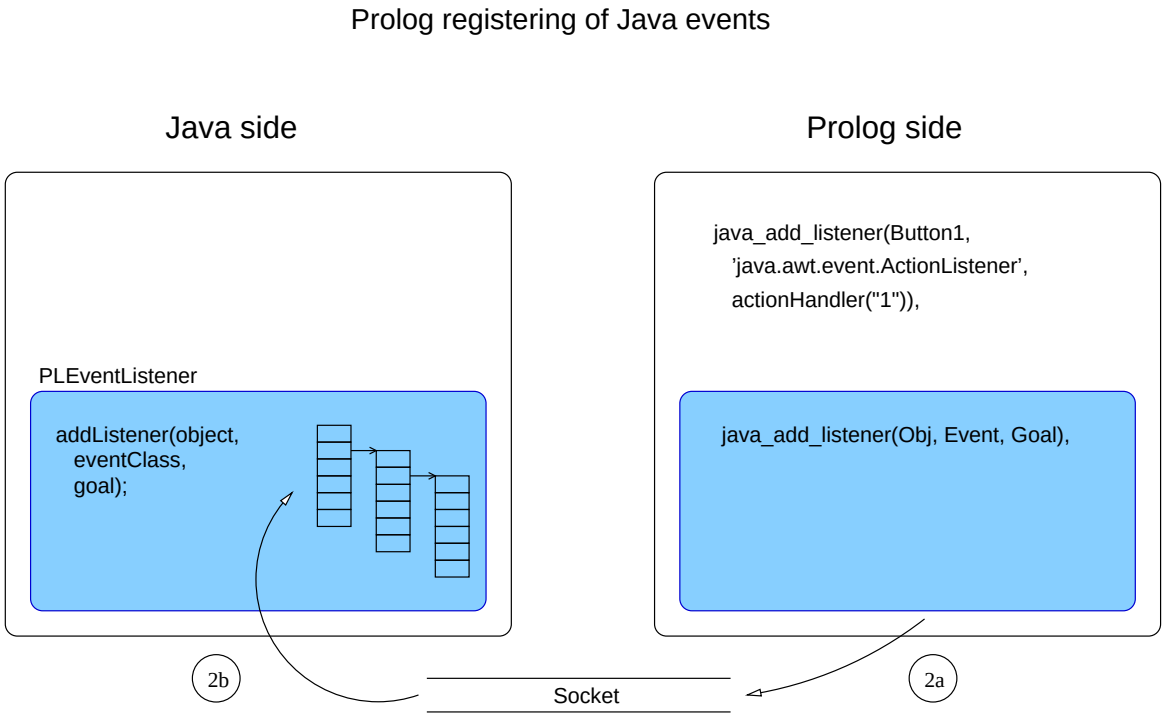
143.2 Java event handling from Prolog

Java event handling is based on a delegation model since version 1.1.x. This approach to event handling is very powerful and elegant, but a user program cannot handle all the events that can arise on a given object: for each kind of event, a listener must be implemented and added specifically. However, the Java 2 API includes a special listener (`AWTEventListener`) that can manage the internal java event queue.

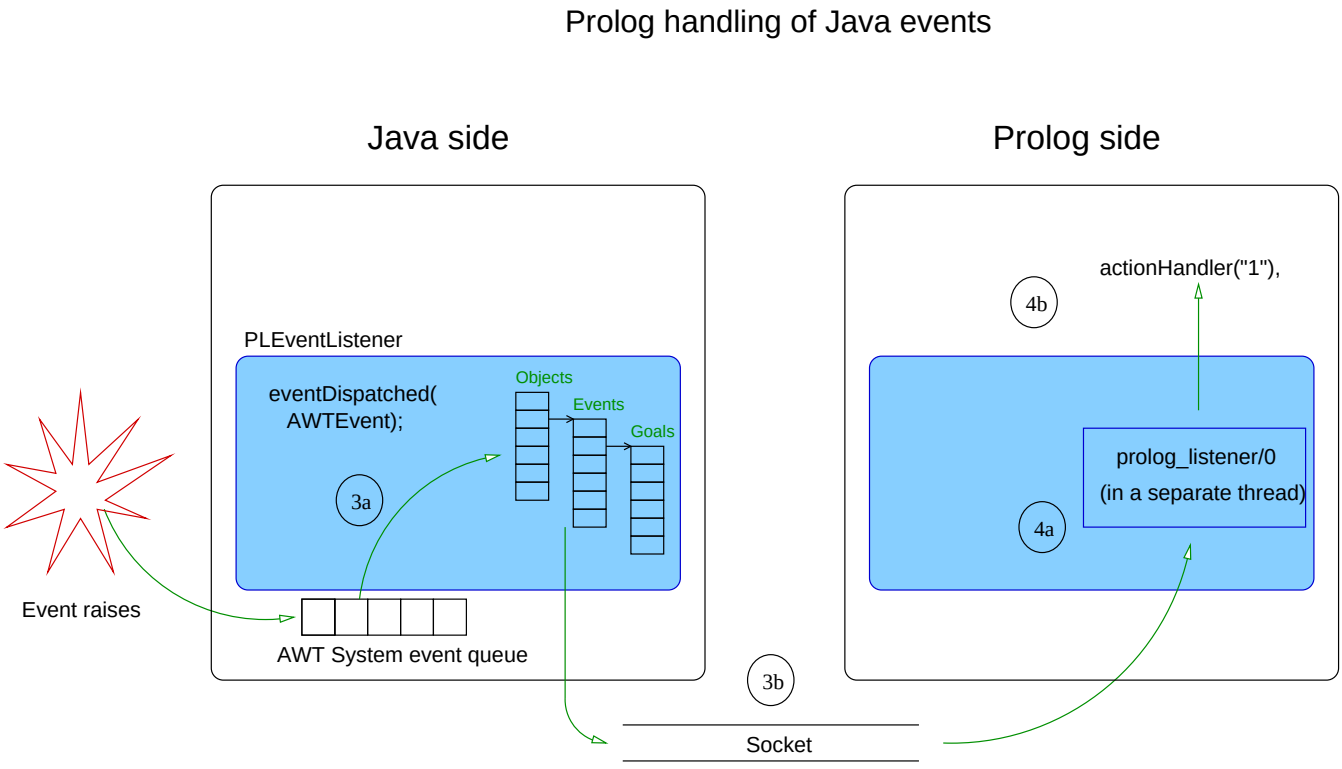
The prolog to java interface has been designed to emulate the java event handler, and is also based on event objects and listeners. The prolog to java interface implements its own event manager, to handle those events that have prolog listeners associated to the object that raises the event. From the prolog side can be added listeners to objects for specific events. The java side includes a list of goals to launch from the object and event type.

Due to the events nature, the event handler must work in a separate thread to manage the events asynchronously. The java side has its own mechanisms to work this way. The prolog side must be implemented specially for event handling using threads. The communication between java and prolog is also asynchronous, and an additional socket stream is used to avoid interferences with the main socket stream. The event stream will work in this implementation only in one way: from java to prolog. If an event handler needs to send back requests to java, it will use the main socket stream, just like the requests sent directly from a prolog program.

The internal process of register a Prolog event handler to a Java event is shown in the next figure:



When an event raises, the Prolog to Java interface has to send to the Prolog user program the goal to evaluate. Graphically, the complete process takes the tasks involved in the following figure:



143.3 Java exception handling from Prolog

Java exception handling is very similar to the peer prolog handling: it includes some specific statements to trap exceptions from user code. In the java side, the exceptions can be originated from an incorrect request, or can be originated in the code called from the request. Both exception types will be sent to prolog using the main socket stream, allowing the prolog program manage the exception. However, the first kind of exceptions are prefixed, so the user program can distinguish them from the second type of exceptions.

In order to handle exceptions properly using the prolog to java and java to prolog interfaces simultaneously, in both sides of the interface those exceptions coming from their own side will be filtered: this avoids an endless loop of exceptions bouncing from one side to another.

143.4 Usage and interface (javart)

- **Library usage:**
  `:- use_module(library(javart)).`
- **Exports:**
  - *Predicates:*
    `java_start/0`, `java_start/1`, `java_start/2`, `java_stop/0`, `java_connect/2`, `java_disconnect/0`, `java_use_module/1`, `java_create_object/2`, `java_delete_object/1`, `java_invoke_method/2`, `java_get_value/2`, `java_set_value/2`, `java_add_listener/3`, `java_remove_listener/3`.
  - *Regular Types:*
    `machine_name/1`, `java_constructor/1`, `java_object/1`, `java_event/1`, `prolog_goal/1`, `java_field/1`, `java_method/1`.
- **Other modules used:**
  - *System library modules:*
    `concurrency/concurrency`, `iso_byte_char`, `lists`, `read`, `write`, `javall/javasock`, `system`.

143.5 Documentation on exports (javart)

java_start/0: PREDICATE

Usage:

- *Description:* Starts the Java server on the local machine, connects to it, and starts the event handling thread.

java_start/1: PREDICATE

Usage: `java_start(Classpath)`

- *Description:* Starts the Java server on the local machine, connects to it, and starts the event handling thread. The Java server is started using the classpath received as argument.
- *Call and exit should be compatible with:*
  `Classpath` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold at call time:*

`Classpath` is currently a term which is not a free variable. (term_typing:nonvar/1)

java_start/2: PREDICATE

Usage: `java_start(Machine_name,Classpath)`

– *Description:* Starts the Java server in `machine_name` (using rsh!), connects to it, and starts the event handling thread. The Java server is started using the `Classpath` received as argument.

– *Call and exit should be compatible with:*

`Machine_name` is the network name of a machine. (javart:machine_name/1)

`Classpath` is a string (a list of character codes). (basic_props:string/1)

– *The following properties should hold at call time:*

`Machine_name` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Classpath` is currently a term which is not a free variable. (term_typing:nonvar/1)

java_stop/0: PREDICATE

Usage:

– *Description:* Stops the interface terminating the threads that handle the socket connection, and finishing the Java interface server if it was started using java_start/n.

java_connect/2: PREDICATE

Usage: `java_connect(Machine_name,Port_number)`

– *Description:* Connects to an existing Java interface server running in `Machine_name` and listening at port `port_number`. To connect to a Java server located in the local machine, use 'localhost' as machine_name.

– *Call and exit should be compatible with:*

`Machine_name` is the network name of a machine. (javart:machine_name/1)

`Port_number` is an integer. (basic_props:int/1)

– *The following properties should hold at call time:*

`Machine_name` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Port_number` is currently a term which is not a free variable. (term_typing:nonvar/1)

java_disconnect/0: PREDICATE

Usage:

– *Description:* Closes the connection with the java process, terminating the threads that handle the connection to Java. This predicate does not terminate the Java process (this is the disconnection procedure for Java servers not started from Prolog). This predicate should be used when the communication is established with java_connect/2.

machine_name/1: REGTYPE

Usage: `machine_name(X)`

– *Description:* `X` is the network name of a machine.

java_constructor/1: REGTYPE

Usage: `java_constructor(X)`

– *Description:* `X` is a java constructor (structure with functor as constructor full name, and arguments as constructor arguments).

java_object/1: REGTYPE

Usage: `java_object(X)`

– *Description:* `X` is a java object (a structure with functor '$java_object', and argument an integer given by the java side).

java_event/1: REGTYPE

Usage: `java_event(X)`

– *Description:* `X` is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener').

prolog_goal/1: REGTYPE

Usage: `prolog_goal(X)`

– *Description:* `X` is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called.

java_field/1: REGTYPE

Usage: `java_field(X)`

– *Description:* `X` is a java field (structure on which the functor name is the field name, and the single argument is the field value).

java_use_module/1: PREDICATE

Usage: `java_use_module(Module)`

– *Description:* Loads a module and makes it available from Java.

– *Call and exit should be compatible with:*

`Module` is any term. (basic_props:term/1)

– *The following properties should hold at call time:*

`Module` is currently a term which is not a free variable. (term_typing:nonvar/1)

java_create_object/2: PREDICATE

Usage:

– *Description:* New java object creation. The constructor must be a compound term as defined by its type, with the full class name as functor (e.g., 'java.lang.String'), and the parameters passed to the constructor as arguments of the structure.

– *Call and exit should be compatible with:*

`Arg1` is a java constructor (structure with functor as constructor full name, and arguments as constructor arguments). (javart:java_constructor/1)

`Arg2` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

– *The following properties should hold at call time:*

`Arg1` is a java constructor (structure with functor as constructor full name, and arguments as constructor arguments). (javart:java_constructor/1)

`Arg2` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Arg2` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

java_delete_object/1: PREDICATE

Usage:

– *Description:* Java object deletion. It removes the object given as argument from the Java object table.

– *Call and exit should be compatible with:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

– *The following properties should hold at call time:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

java_invoke_method/2: PREDICATE

Usage:

– *Description:* Invokes a java method on an object. Given a Java object reference, invokes the method represented with the second argument.

– *Call and exit should be compatible with:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java method (structure with functor as method name, and arguments as method ones, plus a result argument. This result argument is unified with the atom 'Yes' if the java method returns void). (javart:java_method/1)

– *The following properties should hold at call time:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java method (structure with functor as method name, and arguments as method ones, plus a result argument. This result argument is unified with the atom 'Yes' if the java method returns void). (javart:java_method/1)

java_method/1: REGTYPE

Usage: `java_method(X)`

– *Description:* `X` is a java method (structure with functor as method name, and arguments as method ones, plus a result argument. This result argument is unified with the atom 'Yes' if the java method returns void).

java_get_value/2: PREDICATE

Usage:

– *Description:* Gets the value of a field. Given a Java object as first argument, it instantiates the variable given as second argument. This field must be uninstantiated in the java_field functor, or this predicate will fail.

– *Call and exit should be compatible with:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java field (structure on which the functor name is the field name, and the single argument is the field value). (javart:java_field/1)

– *The following properties should hold at call time:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java field (structure on which the functor name is the field name, and the single argument is the field value). (javart:java_field/1)

java_set_value/2: PREDICATE

Usage:

– *Description:* Sets the value of a Java object field. Given a Java object reference, it assigns the value included in the java_field compound term. The field value in the java_field structure must be instantiated.

– *Call and exit should be compatible with:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java field (structure on which the functor name is the field name, and the single argument is the field value). (javart:java_field/1)

– *The following properties should hold at call time:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java field (structure on which the functor name is the field name, and the single argument is the field value). (javart:java_field/1)

java_add_listener/3: PREDICATE

Meta-predicate with arguments: `java_add_listener(?,?,goal)`.

Usage:

– *Description:* Adds a listener to an event on an object. Given a Java object reference, it registers the goal received as third argument to be launched when the Java event raises.

– *Call and exit should be compatible with:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener'). (javart:java_event/1)

`Arg3` is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java

objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called. (javart:prolog_goal/1)

– *The following properties should hold at call time:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener'). (javart:java_event/1)

`Arg3` is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called. (javart:prolog_goal/1)

java_remove_listener/3: PREDICATE

Usage:

– *Description:* It removes a listener from an object event queue. Given a Java object reference, goal registered for the given event is removed.

– *Call and exit should be compatible with:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener'). (javart:java_event/1)

`Arg3` is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called. (javart:prolog_goal/1)

– *The following properties should hold at call time:*

`Arg1` is a java object (a structure with functor '$java_object', and argument an integer given by the java side). (javart:java_object/1)

`Arg2` is a java event represented as an atom with the full event constructor name (e.g., 'java.awt.event.ActionListener'). (javart:java_event/1)

`Arg3` is a prolog predicate. Prolog term that represents the goal that must be invoked when the event raises on the object. The predicate arguments can be java objects, or even the result of java methods. These java objects will be evaluated when the event raises (instead of when the listener is added). The arguments that represent java objects must be instantiated to already created objects. The variables will be kept uninstantiated when the event raises and the predicate is called. (javart:prolog_goal/1)

144 Java to Prolog interface

Author(s): Jesús Correas.

This module defines the Prolog side of the Java to Prolog interface. This side of the interface only has one public predicate: a server that listens at the socket connection with Java, and executes the commands received from the Java side.

In order to evaluate the goals received from the Java side, this module can work in two ways: executing them in the same engine, or starting a thread for each goal. The easiest way is to launch them in the same engine, but the goals must be evaluated sequentially: once a goal provides the first solution, all the subsequent goals must be finished before this goal can backtrack to provide another solution. The Prolog side of this interface works as a top-level, and the goals partially evaluated are not independent.

The solution of this goal dependence is to evaluate the goals in a different prolog engine. Although Ciao includes a mechanism to evaluate goals in different engines, the approach used in this interface is to launch each goal in a different thread.

The decision of what kind of goal evaluation is selected is done by the Java side. Each evaluation type has its own command terms, so the Java side can choose the type it needs.

A Prolog server starts by calling the `prolog_server/0` predicate, or by calling `prolog_server/1` predicate and providing the port number as argument. The user predicates and libraries to be called from Java must be included in the executable file, or be accesible using the built-in predicates dealing with code loading.

144.1 Usage and interface (`jtopl`)

- **Library usage:**
  `:- use_module(library(jtopl)).`
- **Exports:**
  - *Predicates:*
    `prolog_server/0`, `prolog_server/1`, `prolog_server/2`, `shell_s/0`, `query_solutions/2`, `query_requests/2`, `running_queries/2`.
- **Other modules used:**
  - *System library modules:*
    `concurrency/concurrency`, `system`, `read`, `compiler/compiler`, `javall/javasock`.

144.2 Documentation on exports (`jtopl`)

prolog_server/0: PREDICATE

Usage:

- *Description:* Prolog server entry point. Reads from the standard input the node name and port number where the java client resides, and starts the prolog server listening at the jp socket. This predicate acts as a server: it includes an endless read-process loop until the `prolog_halt` command is received.

  However, from the low-level communication point of view, this Prolog server actually works as a client of the Java side. This means that Java side waits at the given port to a Prolog server trying to create a socket; Prolog side connects to that port, and then waits for Java requests (acting as a 'logical' server). To use this Prolog server as a real server waiting for connections at a given port, use `prolog_server/1`.

prolog_server/1: PREDICATE

Usage:

– *Description:* Waits for incoming Java connections to act as a Prolog goal server for Java requests.This is the only `prolog_server/*` predicate that works as a true server: given a port number, waits for a connection from Java and then serves Java requests. When a termination request is received, finishes the connection to Java and waits next Java connection request. This behaviour is different with respect to previous versions of this library. To work as before, use `prolog_server/2`.

Although it currently does not support simultaneous Java connections, some work is being done in that direction.

– *Call and exit should be compatible with:*

`Arg1` is an atom. (basic_props:atm/1)

prolog_server/2: PREDICATE

Usage:

– *Description:* Prolog server entry point. Given a network `node` and a `port` number, starts the prolog server trying to connect to Java side at that `node:port` address, and then waits for Java requests. This predicate acts as a server: it includes an endless read-process loop until the `prolog_halt` command is received.

However, from the low-level communication point of view, this Prolog server actually works as a client of the Java side. This means that Java side waits at the given port to a Prolog server trying to create a socket; Prolog side connects to that port, and then waits for Java requests (acting as a 'logical' server). To use this Prolog server as a real server waiting for connections at a given port, use `prolog_server/1`.

– *Call and exit should be compatible with:*

`Arg1` is an atom. (basic_props:atm/1)

`Arg2` is an atom. (basic_props:atm/1)

shell_s/0: PREDICATE

Usage:

– *Description:* Command execution loop. This predicate is called when the connection to Java is established, and performs an endless loop processing the commands received. This predicate is only intended to be used by the Prolog to Java interface and it should not be used by a user program.

query_solutions/2: PREDICATE

No further documentation available for this predicate.

The predicate is of type *concurrent*.

query_requests/2: PREDICATE

No further documentation available for this predicate.

The predicate is of type *concurrent*.

running_queries/2: PREDICATE

No further documentation available for this predicate.

The predicate is of type *concurrent*.

145 Low-level Prolog to Java socket connection

Author(s): Jesús Correas.

This module defines a low-level socket interface, to be used by javart and jtopl. Includes all the code related directly to the handling of sockets. This library should not be used by any user program, because is a very low-level connection to Java. Use `javart` (Prolog to Java interface) or `jtopl` (Java to Prolog interface) libraries instead.

145.1 Usage and interface (`javasock`)

- **Library usage:**
  `:- use_module(library(javasock)).`
- **Exports:**
  - *Predicates:*
    `bind_socket_interface/1`, `start_socket_interface/2`, `stop_socket_interface/0`, `join_socket_interface/0`, `java_query/2`, `java_response/2`, `prolog_query/2`, `prolog_response/2`, `is_connected_to_java/0`, `java_debug/1`, `java_debug_redo/1`, `start_threads/0`.
- **Other modules used:**
  - *System library modules:*
    `fastrw`, `sockets/sockets`, `format`, `concurrency/concurrency`, `javall/jtopl`, `sockets/sockets_io`.

145.2 Documentation on exports (`javasock`)

bind_socket_interface/1: PREDICATE

Usage: `bind_socket_interface(Port)`

- *Description:* Given an port number, waits for a connection request from the Java side, creates the sockets to connect to the java process, and starts the threads needed to handle the connection.
- *Call and exit should be compatible with:*
  `Port` is an integer. (basic_props:int/1)
- *The following properties should hold at call time:*
  `Port` is currently a term which is not a free variable. (term_typing:nonvar/1)

start_socket_interface/2: PREDICATE

Usage: `start_socket_interface(Address,Stream)`

- *Description:* Given an address in format 'node:port', creates the sockets to connect to the java process, and starts the threads needed to handle the connection.
- *Call and exit should be compatible with:*
  `Address` is any term. (basic_props:term/1)
  `Stream` is an open stream. (streams_basic:stream/1)

– *The following properties should hold at call time:*

`Address` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Stream` is currently a term which is not a free variable. (term_typing:nonvar/1)

stop_socket_interface/0: PREDICATE

Usage:

– *Description:* Closes the sockets to disconnect from the java process, and waits until the threads that handle the connection terminate.

join_socket_interface/0: PREDICATE

Usage:

– *Description:* Waits until the threads that handle the connection terminate.

java_query/2: PREDICATE

The predicate is of type *concurrent*.

Usage: `java_query(ThreadId,Query)`

– *Description:* Data predicate containing the queries to be sent to Java. First argument is the Prolog thread Id, and second argument is the query to send to Java.

– *Call and exit should be compatible with:*

`ThreadId` is an atom. (basic_props:atm/1)

`Query` is any term. (basic_props:term/1)

java_response/2: PREDICATE

The predicate is of type *concurrent*.

Usage: `java_response(Id,Response)`

– *Description:* Data predicate that stores the responses to requests received from Java. First argument corresponds to the Prolog thread Id; second argument corresponds to the response itself.

– *Call and exit should be compatible with:*

`Id` is an atom. (basic_props:atm/1)

`Response` is any term. (basic_props:term/1)

prolog_query/2: PREDICATE

The predicate is of type *concurrent*.

Usage: `prolog_query(Id,Query)`

– *Description:* Data predicate that keeps a queue of the queries requested to Prolog side from Java side.

– *Call and exit should be compatible with:*

`Id` is an integer. (basic_props:int/1)

`Query` is any term. (basic_props:term/1)

prolog_response/2: PREDICATE

The predicate is of type *concurrent*.

Usage: `prolog_response(Id,Response)`

– *Description:* Data predicate that keeps a queue of the responses to queries requested to Prolog side from Java side.

– *Call and exit should be compatible with:*

`Id` is an integer. (basic_props:int/1)

`Response` is any term. (basic_props:term/1)

is_connected_to_java/0: PREDICATE

Usage:

– *Description:* Checks if the connection to Java is established.

java_debug/1: PREDICATE

No further documentation available for this predicate.

java_debug_redo/1: PREDICATE

No further documentation available for this predicate.

start_threads/0: PREDICATE

Usage:

– *Description:* Starts the threads that will handle the connection to Java. This predicate is declared public for internal purposes, and it is not intended to be used by a user program.

146 Calling emacs from Prolog

Author(s): The CLIP Group.

This library provides a *prolog-emacs interface*. This interface is complementary to (and independent from) the emacs mode, which is used to develop programs from within the `emacs` editor/environment. Instead, this library allows calling `emacs` from a running Prolog program. This facilitates the use of `emacs` as a "user interface" for a Prolog program. Emacs can be made to:

- Visit a file, which can then be edited.
- Execute arbitrary *emacs lisp* code, sent from Prolog.

In order for this library to work correctly, the following is needed:

- You should be running the `emacs` editor on the same machine where the executable calling this library is executing.
- This `emacs` should be running the *emacs server*. This can be done by including the following line in your `.emacs` file:

```
;; Start a server that emacsclient can connect to.
(server-start)
```

  Or typing `M-x server-start` within `emacs`.

This suffices for using `emacs` to edit files. For running arbitrary code the following also needs to be added to the `.emacs` file:

`(setq enable-local-eval t)`
: Allows executing lisp code without asking.

`(setq enable-local-eval nil)`
: Does not allow executing lisp code without asking.

`(setq enable-local-eval 'maybe)`
: Allows executing lisp code only if user agrees after asking (asks interactively for every invocation).

Examples:

Assuming that a `.pl` file loads this library, then:

`..., emacs_edit('foo'), ...`
: Opens file `foo` for editing in `emacs`.

`..., emacs_eval_nowait("(run-ciao-toplevel)"), ...`
: Starts execution of a Ciao top-level within `emacs`.

146.1 Usage and interface (emacs)

- **Library usage:**
  `:- use_module(library(emacs)).`
- **Exports:**
  - *Predicates:*
    `emacs_edit/1`, `emacs_edit_nowait/1`, `emacs_eval/1`, `emacs_eval_nowait/1`.
  - *Regular Types:*
    `elisp_string/1`.
- **Other modules used:**
  - *System library modules:*
    `terms_check`, `lists`, `terms`, `system`.

146.2 Documentation on exports (emacs)

emacs_edit/1: PREDICATE

Usage:

- *Description:* Opens the given file for editing in `emacs`. Waits for editing to finish before continuing.
- *The following properties should hold at call time:*
  `Arg1` is an atom which is the name of a file. (emacs:filename/1)

emacs_edit_nowait/1: PREDICATE

Usage:

- *Description:* Opens the given file for editing in `emacs` and continues without waiting for editing to finish.
- *The following properties should hold at call time:*
  `Arg1` is an atom which is the name of a file. (emacs:filename/1)

emacs_eval/1: PREDICATE

Usage:

- *Description:* Executes in emacs the lisp code given as argument. Waits for the command to finish before continuing.
- *The following properties should hold at call time:*
  `Arg1` is a string containing `emacs` lisp code. (emacs:elisp_string/1)

emacs_eval_nowait/1: PREDICATE

Usage:

- *Description:* Executes in emacs the lisp code given as argument and continues without waiting for it to finish.

– *The following properties should hold at call time:*
`Arg1` is a string containing `emacs` lisp code. (emacs:elisp_string/1)

elisp_string/1: REGTYPE

Usage: `elisp_string(L)`

– *Description:* `L` is a string containing `emacs` lisp code.

147 linda (library)

This is a SICStus-like linda package. Note that this is essentially quite obsolete, and provided mostly in case it is needed for compatibility, since Ciao now supports all Linda functionality (and more) through the concurrent fact database.

147.1 Usage and interface (linda)

- **Library usage:**
  `:- use_module(library(linda)).`
- **Exports:**
  - *Predicates:*
    `linda_client/1`, `close_client/0`, `in/1`, `in/2`, `in_noblock/1`, `out/1`, `rd/1`, `rd/2`, `rd_noblock/1`, `rd_findall/3`, `linda_timeout/2`, `halt_server/0`, `open_client/2`, `in_stream/2`, `out_stream/2`.
- **Other modules used:**
  - *System library modules:*
    `read`, `fastrw`, `sockets/sockets`.

147.2 Documentation on exports (linda)

linda_client/1: PREDICATE

No further documentation available for this predicate.

close_client/0: PREDICATE

No further documentation available for this predicate.

in/1: PREDICATE

No further documentation available for this predicate.

in/2: PREDICATE

No further documentation available for this predicate.

in_noblock/1: PREDICATE

No further documentation available for this predicate.

out/1: PREDICATE

No further documentation available for this predicate.

rd/1: PREDICATE

No further documentation available for this predicate.

rd/2: PREDICATE

No further documentation available for this predicate.

rd_noblock/1: PREDICATE

No further documentation available for this predicate.

rd_findall/3: PREDICATE

No further documentation available for this predicate.

linda_timeout/2: PREDICATE

No further documentation available for this predicate.

halt_server/0: PREDICATE

No further documentation available for this predicate.

open_client/2: PREDICATE

No further documentation available for this predicate.

in_stream/2: PREDICATE

No further documentation available for this predicate.

out_stream/2: PREDICATE

No further documentation available for this predicate.

PART IX - Abstract data types

Author(s): The CLIP Group.

This part includes libraries which implement some generic data structures (abstract data types) that are used frequently in programs or in the Ciao system itself.

148 Extendable arrays with logarithmic access time

Author(s): Lena Flood.

This module implements extendable arrays with logarithmic access time. It has been adapted from shared code written by David Warren and Fernando Pereira.

148.1 Usage and interface (arrays)

- **Library usage:**
  `:- use_module(library(arrays)).`
- **Exports:**
  - *Predicates:*
    `new_array/1`, `is_array/1`, `aref/3`, `arefa/3`, `arefl/3`, `aset/4`, `array_to_list/2`.

148.2 Documentation on exports (arrays)

new_array/1: PREDICATE

Usage: `new_array(Array)`

- *Description:* returns an empty new array `Array`.
- *The following properties should hold at call time:*
  `Array` is a free variable. (term_typing:var/1)

is_array/1: PREDICATE

Usage: `is_array(Array)`

- *Description:* `Array` actually is an array.
- *The following properties should hold at call time:*
  `Array` is currently a term which is not a free variable. (term_typing:nonvar/1)

aref/3: PREDICATE

Usage: `aref(Index,Array,Element)`

- *Description:* unifies `Element` to `Array`[`Index`], or fails if `Array`[`Index`] has not been set.
- *The following properties should hold at call time:*
  `Index` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Array` is currently a term which is not a free variable. (term_typing:nonvar/1)

arefa/3: PREDICATE

Usage: `arefa(Index,Array,Element)`

- *Description:* is as `aref/3`, except that it unifies `Element` with a new array if `Array`[`Index`] is undefined. This is useful for multidimensional arrays implemented as arrays of arrays.

– *The following properties should hold at call time:*

`Index` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Array` is currently a term which is not a free variable. (term_typing:nonvar/1)

arefl/3: PREDICATE

Usage: `arefl(Index,Array,Element)`

– *Description:* is as `aref/3`, except that `Element` appears as `[]` for undefined cells. Thus, `arefl(_,_,[])` always succeeds no matter what you give in the first or second args.

– *The following properties should hold at call time:*

`Index` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Array` is currently a term which is not a free variable. (term_typing:nonvar/1)

aset/4: PREDICATE

Usage: `aset(Index,Array,Element,NewArray)`

– *Description:* unifies `NewArray` with the result of setting `Array`[`Index`] to `Element`.

– *The following properties should hold at call time:*

`Index` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Array` is currently a term which is not a free variable. (term_typing:nonvar/1)

`NewArray` is a free variable. (term_typing:var/1)

array_to_list/2: PREDICATE

Usage: `array_to_list(Array,List)`

– *Description:* returns a `List` of pairs Index-Element of all the elements of `Array` that have been set.

– *The following properties should hold at call time:*

`Array` is currently a term which is not a free variable. (term_typing:nonvar/1)

`List` is a free variable. (term_typing:var/1)

149 Association between key and value

Author(s): Pablo Chico, Manuel Carro.

This library implements a table. It takes its name from the classical "association lists". It allows storing a set of values and a key for each value, such that the values can later be accessed through these keys. These keys could not be ground terms, but they could not be instanciated later (so, this implementation unify with '==' instead of '='). The implementation uses a dynamically changing data structure for efficiency. When there are few elements the data structure used is a list of pairs. When the number of elements stored goes beyond some number, an AVL tree is used. There is a certain level of hysteresis so that no repeated data structure conversions occur when the number of elements is close to the threshold.

149.1 Usage and interface (assoc)

- **Library usage:**
  `:- use_module(library(assoc)).`
- **Exports:**
  - *Predicates:*
    `empty_assoc/1`, `assoc_to_list/2`, `is_assoc/1`, `min_assoc/3`, `max_assoc/3`, `gen_assoc/3`, `get_assoc/3`, `get_assoc/5`, `get_next_assoc/4`, `get_prev_assoc/4`, `list_to_assoc/2`, `ord_list_to_assoc/2`, `map_assoc/2`, `map_assoc/3`, `map/3`, `foldl/4`, `put_assoc/4`, `put_assoc/5`, `add_assoc/4`, `update_assoc/5`, `del_assoc/4`, `del_min_assoc/4`, `del_max_assoc/4`.
- **Other modules used:**
  - *System library modules:*
    `hiordlib`, `lists`.

149.2 Documentation on exports (assoc)

empty_assoc/1: PREDICATE

Usage 1: `empty_assoc(Assoc)`

- *Description:* True if `Assoc` is an empty `assoc_table`.
- *The following properties should hold at call time:*
  `Assoc` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Assoc` is a associations beetwen keys and values. (assoc:assoc_table/1)

Usage 2: `empty_assoc(Assoc)`

- *Description:* `Assoc` is an empty `assoc_table`.
- *The following properties should hold at call time:*
  `Assoc` is a free variable. (term_typing:var/1)
  `Assoc` is a associations beetwen keys and values. (assoc:assoc_table/1)

assoc_to_list/2: PREDICATE

Usage: `assoc_to_list(Assoc,L)`

– *Description:* Transforms `Assoc` into `L` where each pair of `L` was a association in `Assoc`.
– *The following properties should hold at call time:*
`Assoc` is currently a term which is not a free variable. (term_typing:nonvar/1)
`L` is a free variable. (term_typing:var/1)
`Assoc` is a associations beetwen keys and values. (assoc:assoc_table/1)
`L` is a ordered list of elements of the form `key-value`. (assoc:ord_pairs/1)

is_assoc/1: PREDICATE

Usage: `is_assoc(Assoc)`

– *Description:* True if `Assoc` is an `assoc_table`.
– *The following properties should hold at call time:*
`Assoc` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Assoc` is a associations beetwen keys and values. (assoc:assoc_table/1)

min_assoc/3: PREDICATE

Usage: `min_assoc(Assoc,Key,Value)`

– *Description:* `Key` and `Value` are `key` and `value` of the element with the smallest `key` in `Assoc`.
– *The following properties should hold at call time:*
`Assoc` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Key` is a free variable. (term_typing:var/1)
`Value` is a free variable. (term_typing:var/1)
`Assoc` is a associations beetwen keys and values. (assoc:assoc_table/1)
`K` is a valid key in a `assoc_table`. (assoc:key/1)
`Value` is a valid value in a `assoc_table`. (assoc:value/1)

max_assoc/3: PREDICATE

Usage: `max_assoc(Assoc,Key,Value)`

– *Description:* `Key` and `Value` are the `key` and `value` of the element with the largest `key` in `Assoc`.
– *The following properties should hold at call time:*
`Assoc` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Key` is a free variable. (term_typing:var/1)
`Value` is a free variable. (term_typing:var/1)
`Assoc` is a associations beetwen keys and values. (assoc:assoc_table/1)
`Key` is a valid key in a `assoc_table`. (assoc:key/1)
`Value` is a valid value in a `assoc_table`. (assoc:value/1)

gen_assoc/3: PREDICATE

Usage 1: `gen_assoc(K,Assoc,V)`

– *Description:* Enumerate matching elements of Assoc in ascending order of their keys via backtracking.
– *The following properties should hold at call time:*

`K` is a free variable.	(term_typing:var/1)
`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`V` is a free variable.	(term_typing:var/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`.	(assoc:value/1)

Usage 2: `gen_assoc(K,Assoc,V)`

– *Description:* Enumerate matching elements of Assoc in ascending order of their keys via backtracking whose value is `V`.
– *The following properties should hold at call time:*

`K` is a free variable.	(term_typing:var/1)
`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`V` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`.	(assoc:value/1)

get_assoc/3: PREDICATE

Usage 1: `get_assoc(K,Assoc,V)`

– *Description:* True if `V` is the value associated to the key `K` in the assoc_table `Assoc`.
– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`V` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`.	(assoc:value/1)

Usage 2: `get_assoc(K,Assoc,V)`

– *Description:* `V` is the value associated to the key `K` in the assoc_table `Assoc`.
– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`V` is a free variable.	(term_typing:var/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`.	(assoc:value/1)

get_assoc/5: PREDICATE

Usage: `get_assoc(K,Assoc,Old,NewAssoc,New)`

– *Description:* `NewAssoc` is an `assoc_table` identical to `Assoc` except that the value associated with `Key` is `New` instead of `Old`.

– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Old` is a free variable.	(term_typing:var/1)
`NewAssoc` is a free variable.	(term_typing:var/1)
`New` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`Old` is a valid value in a `assoc_table`.	(assoc:value/1)
`NewAssoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`New` is a valid value in a `assoc_table`.	(assoc:value/1)

get_next_assoc/4: PREDICATE

Usage: `get_next_assoc(K,Assoc,NextK,NextV)`

– *Description:* `NextK` and `NextV` are the next `key` and associated `value` after `K` in `Assoc`.

– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`NextK` is a free variable.	(term_typing:var/1)
`NextV` is a free variable.	(term_typing:var/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`NextK` is a valid key in a `assoc_table`.	(assoc:key/1)
`NextV` is a valid value in a `assoc_table`.	(assoc:value/1)

get_prev_assoc/4: PREDICATE

Usage: `get_prev_assoc(K,Assoc,PrevK,PrevV)`

– *Description:* `PrevK` and `PrevV` are the previous `key` and associated `value` after `K` in `Assoc`.

– *The following properties should hold at call time:*

`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`PrevK` is a valid key in a `assoc_table`.	(assoc:key/1)
`PrevV` is a valid value in a `assoc_table`.	(assoc:value/1)

list_to_assoc/2: PREDICATE

Usage: `list_to_assoc(L,Assoc)`

– *Description:* Transforms `L` into `Assoc` where each pair of `L` will be a association in `Assoc`.
– *The following properties should hold at call time:*

`L` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc` is a free variable.	(term_typing:var/1)
`L` is a list of elements of the form `key-value`.	(assoc:pairs/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)

ord_list_to_assoc/2: PREDICATE

Usage: `ord_list_to_assoc(L,Assoc)`

– *Description:* Transforms `L`, a list of pairs (using the functor `-/2`) sorted by its first element, into the table `Assoc` where each pair of `L` will become a association in `Assoc`.
– *The following properties should hold at call time:*

`L` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc` is a free variable.	(term_typing:var/1)
`L` is a ordered list of elements of the form `key-value`.	(assoc:ord_pairs/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)

map_assoc/2: PREDICATE

Meta-predicate with arguments: `map_assoc((pred 1),?)`.

Usage: `map_assoc(Pred,Assoc)`

– *Description:* `Assoc` is an association tree, and for each `Key`, if `Key` is associated with `Value` in `Assoc`, Pred(Value) is true.
– *The following properties should hold at call time:*

`Pred` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)

map_assoc/3: PREDICATE

Meta-predicate with arguments: `map_assoc((pred 2),?,?)`.

Usage: `map_assoc(Pred,Assoc,NewAssoc)`

– *Description:* `Assoc` and `NewAssoc` are association trees of the same shape, and for each `Key`, if `Key` is associated with `Old` in `Assoc` and with `new` in `NewAssoc`, Pred(Old,New) is true.
– *The following properties should hold at call time:*

`Pred` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`NewAssoc` is a free variable.	(term_typing:var/1)

map/3: PREDICATE

Meta-predicate with arguments: `map(?,(pred 3),?)`.

Usage: `map(Assoc1,Pred,Assoc2)`

– *Description:* Applies `Pred` with arity 3 to each value of the assoc_table `Assoc1` obtaining the new assoc_table `Assoc2` in which only the values can have changed.
– *The following properties should hold at call time:*

`Assoc1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Pred` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Assoc2` is a free variable. (term_typing:var/1)

foldl/4: PREDICATE

Meta-predicate with arguments: `foldl(?,?,(pred 4),?)`.

Usage: `foldl(Assoc,DS,Pred,NDS)`

– *Description:* Applies `Pred` with arity 4 to each value of the assoc_table `Assoc`. If `Pred` is satisfied, it updates the data-structure DS. Otherwise it fails.
– *The following properties should hold at call time:*

`Assoc` is currently a term which is not a free variable. (term_typing:nonvar/1)
`DS` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Pred` is currently a term which is not a free variable. (term_typing:nonvar/1)
`NDS` is a free variable. (term_typing:var/1)

put_assoc/4: PREDICATE

Usage: `put_assoc(K,Assoc,V,NewAssoc)`

– *Description:* The value `V` is inserted in `Assoc` associated to the key `K` and the result is `NewAssoc`. This can be used to insert and change associations.
– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Assoc` is currently a term which is not a free variable. (term_typing:nonvar/1)
`V` is currently a term which is not a free variable. (term_typing:nonvar/1)
`NewAssoc` is a free variable. (term_typing:var/1)
`K` is a valid key in a `assoc_table`. (assoc:key/1)
`Assoc` is a associations beetwen keys and values. (assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`. (assoc:value/1)
`NewAssoc` is a associations beetwen keys and values. (assoc:assoc_table/1)

put_assoc/5: PREDICATE

Usage: `put_assoc(K,Assoc1,V,Assoc2,Member)`

– *Description:* The value `V` is inserted in `Assoc1` associated to the key `K` and the result is `Assoc2`. If the key `K` doesn't belong to the `Assoc1` then `Member` is unified with no. Otherwise, `Assoc2` is the result of substituting the association `K-OldValue` by `K-V` and `Member` is unified with yes(OldValue).
– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Assoc1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`V` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Assoc2` is a free variable. (term_typing:var/1)
`Member` is a free variable. (term_typing:var/1)
`K` is a valid key in a `assoc_table`. (assoc:key/1)
`Assoc1` is a associations beetwen keys and values. (assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`. (assoc:value/1)
`Assoc2` is a associations beetwen keys and values. (assoc:assoc_table/1)
`Member` is no or yes(value). (assoc:is_member/1)

add_assoc/4: PREDICATE

Usage: `add_assoc(K,Assoc1,V,Assoc2)`

– *Description:* This is similar to `put_value/5` but `Key` must not appear in `Assoc1` (`Member` in put_value/5 is known to be no). An error is thrown otherwise.
– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Assoc1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`V` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Assoc2` is a free variable. (term_typing:var/1)
`K` is a valid key in a `assoc_table`. (assoc:key/1)
`Assoc1` is a associations beetwen keys and values. (assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`. (assoc:value/1)
`Assoc2` is a associations beetwen keys and values. (assoc:assoc_table/1)

update_assoc/5: PREDICATE

Usage: `update_assoc(K,Assoc1,V,Assoc2,OldVar)`

– *Description:* This is similar to `put_assoc/5` but `Key` must not appear in `Assoc1` (`Member` in put_value/5 is known to be no). An error is thrown otherwise.
– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Assoc1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`V` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Assoc2` is a free variable. (term_typing:var/1)
`OldVar` is a free variable. (term_typing:var/1)
`K` is a valid key in a `assoc_table`. (assoc:key/1)
`Assoc1` is a associations beetwen keys and values. (assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`. (assoc:value/1)
`Assoc2` is a associations beetwen keys and values. (assoc:assoc_table/1)
`OldVar` is a valid value in a `assoc_table`. (assoc:value/1)

del_assoc/4: PREDICATE

Usage: `del_assoc(K,Assoc1,V,Assoc2)`

– *Description:* Delete in `Assoc1` the key `K` to give `Assoc2`. If the key `K` does not belong to the `Assoc1` then `Member` is unified with no and `Assoc1` and `Assoc2` are unified. Otherwise `Assoc2` is the result of deleting the key `K` and its associated `Value`, and `Member` is unified with yes(Value).
– *The following properties should hold at call time:*

`K` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Assoc1` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`V` is a free variable.	(term_typing:var/1)
`Assoc2` is a free variable.	(term_typing:var/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`Assoc1` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`V` is a valid value in a `assoc_table`.	(assoc:value/1)
`Assoc2` is a associations beetwen keys and values.	(assoc:assoc_table/1)

del_min_assoc/4: PREDICATE

Usage: `del_min_assoc(Assoc,K,V,NewAssoc)`

– *Description:* `Assoc` and `NewAssoc` define the same finite function except that `Assoc` associates `K` with `V` and `NewAssoc` doesn't associate `K` with any value and `K` precedes all other keys in `Assoc`.
– *The following properties should hold at call time:*

`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`K` is a free variable.	(term_typing:var/1)
`V` is a free variable.	(term_typing:var/1)
`NewAssoc` is a free variable.	(term_typing:var/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`V` is a valid value in a `assoc_table`.	(assoc:value/1)
`NewAssoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)

del_max_assoc/4: PREDICATE

Usage: `del_max_assoc(Assoc,K,V,NewAssoc)`

– *Description:* `Assoc` and `NewAssoc` define the same finite function except that `Assoc` associates `K` with `V` and `NewAssoc` doesn't associate `K` with any value and `K` is preceded by all other keys in `Assoc`.
– *The following properties should hold at call time:*

`Assoc` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`K` is a free variable.	(term_typing:var/1)
`V` is a free variable.	(term_typing:var/1)
`NewAssoc` is a free variable.	(term_typing:var/1)
`Assoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)
`K` is a valid key in a `assoc_table`.	(assoc:key/1)
`V` is a valid value in a `assoc_table`.	(assoc:value/1)
`NewAssoc` is a associations beetwen keys and values.	(assoc:assoc_table/1)

150 counters (library)

150.1 Usage and interface (counters)

- **Library usage:**
  `:- use_module(library(counters)).`
- **Exports:**
  - *Predicates:*
    `setcounter/2`, `getcounter/2`, `inccounter/2`.

150.2 Documentation on exports (counters)

setcounter/2: PREDICATE

No further documentation available for this predicate.

getcounter/2: PREDICATE

No further documentation available for this predicate.

inccounter/2: PREDICATE

No further documentation available for this predicate.

151 Identity lists

Author(s): Francisco Bueno.

The operations in this module handle lists by performing equality checks via identity instead of unification.

151.1 Usage and interface (idlists)

- **Library usage:**
  `:- use_module(library(idlists)).`
- **Exports:**
  - *Predicates:*
    `member_0/2`, `memberchk/2`, `list_insert/2`, `add_after/4`, `add_before/4`, `delete/3`, `subtract/3`, `union_idlists/3`.

151.2 Documentation on exports (idlists)

member_0/2: PREDICATE

`member_0(X,Xs)`

True iff memberchk/2 is true.

memberchk/2: PREDICATE

`memberchk(X,Xs)`

Checks that `X` is an element of (list) `Xs`.

list_insert/2: PREDICATE

Usage: `list_insert(List,Term)`

- *Description:* Adds `Term` to the end of (tail-opened) `List` if there is not an element in `List` identical to `Term`.
- *The following properties should hold at call time:*
  `List` is a free variable. (term_typing:var/1)
  `Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

add_after/4: PREDICATE

Usage: `add_after(L0,E0,E,L)`

- *Description:* Adds element `E` after the first element identical to `E0` (or at end) of list `L0`, returning in `L` the new list.
- *The following properties should hold at call time:*
  `L0` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `E0` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `E` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `L` is a free variable. (term_typing:var/1)

add_before/4: PREDICATE

Usage: `add_before(L0,E0,E,L)`

– *Description:* Adds element `E` before the first element identical to `E0` (or at start) of list `L0`, returning in `L` the new list.

– *The following properties should hold at call time:*

`L0` is currently a term which is not a free variable. (term_typing:nonvar/1)

`E0` is currently a term which is not a free variable. (term_typing:nonvar/1)

`E` is currently a term which is not a free variable. (term_typing:nonvar/1)

`L` is a free variable. (term_typing:var/1)

delete/3: PREDICATE

Usage: `delete(List,Element,Rest)`

– *Description:* `Rest` has the same elements of `List` except for all the occurrences of elements identical to `Element`.

– *The following properties should hold at call time:*

`List` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Element` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Rest` is a free variable. (term_typing:var/1)

subtract/3: PREDICATE

Usage: `subtract(Set,Set0,Difference)`

– *Description:* `Difference` has the same elements of `Set` except those which have an identical occurrence in `Set0`.

– *The following properties should hold at call time:*

`Set` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Set0` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Difference` is a free variable. (term_typing:var/1)

union_idlists/3: PREDICATE

Usage: `union_idlists(List1,List2,List)`

– *Description:* `List` has the elements which are in `List1` but are not identical to an element in `List2` followed by the elements in `List2`.

– *The following properties should hold at call time:*

`List1` is currently a term which is not a free variable. (term_typing:nonvar/1)

`List2` is currently a term which is not a free variable. (term_typing:nonvar/1)

`List` is a free variable. (term_typing:var/1)

152 Lists of numbers

Author(s): The CLIP Group.
This module implements some kinds of lists of numbers.

152.1 Usage and interface (numlists)

- **Library usage:**
  `:- use_module(library(numlists)).`
- **Exports:**
  - *Predicates:*
    `get_primes/2`, `sum_list/2`, `sum_list/3`, `sum_list_of_lists/2`, `sum_list_of_lists/3`.
  - *Regular Types:*
    `intlist/1`, `numlist/1`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

152.2 Documentation on exports (numlists)

get_primes/2: PREDICATE

Usage: `get_primes(N,Primes)`

- *Description:* Computes the `N`th first prime numbers in ascending order.
- *The following properties should hold at call time:*
  `N` is an integer. (basic_props:int/1)
- *The following properties should hold upon exit:*
  `Primes` is a list of integers. (numlists:intlist/1)

intlist/1: REGTYPE

Usage: `intlist(X)`

- *Description:* `X` is a list of integers.

numlist/1: REGTYPE

Usage: `numlist(X)`

- *Description:* `X` is a list of numbers.

sum_list/2: PREDICATE

Usage: `sum_list(List,N)`

- *Description:* `N` is the total sum of the elements of `List`.

– *The following properties should hold at call time:*
`List` is a list of numbers. (numlists:numlist/1)
– *The following properties should hold upon exit:*
`N` is a number. (basic_props:num/1)

sum_list/3: PREDICATE
Usage: `sum_list(List,N0,N)`
– *Description:* `N` is the total sum of the elements of `List` plus `N0`.
– *The following properties should hold at call time:*
`List` is a list of numbers. (numlists:numlist/1)
`N0` is a number. (basic_props:num/1)
– *The following properties should hold upon exit:*
`N` is a number. (basic_props:num/1)

sum_list_of_lists/2: PREDICATE
Usage: `sum_list_of_lists(Lists,N)`
– *Description:* `N` is the total sum of the elements of the lists of `Lists`.
– *The following properties should hold at call time:*
`List` is a list of `numlists`. (basic_props:list/2)
– *The following properties should hold upon exit:*
`N` is a number. (basic_props:num/1)

sum_list_of_lists/3: PREDICATE
Usage: `sum_list_of_lists(Lists,N0,N)`
– *Description:* `N` is the total sum of the elements of the lists of `Lists` plus `N0`.
– *The following properties should hold at call time:*
`List` is a list of `numlists`. (basic_props:list/2)
`N0` is a number. (basic_props:num/1)
– *The following properties should hold upon exit:*
`N` is a number. (basic_props:num/1)

153 Pattern (regular expression) matching - deprecated version

Author(s): The CLIP Group.

(Deprecated - please use the new "regexp" package instead.)

This library provides facilities for matching strings and terms against *patterns* (i.e., *regular expressions*).

153.1 Usage and interface (patterns)

- **Library usage:**
  `:- use_module(library(patterns)).`
- **Exports:**
  - *Predicates:*
    `match_pattern/2`, `match_pattern/3`, `case_insensitive_match/2`, `letter_match/2`, `match_pattern_pred/2`.
  - *Regular Types:*
    `pattern/1`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

153.2 Documentation on exports (patterns)

match_pattern/2: PREDICATE

Usage: `match_pattern(Pattern,String)`

- *Description:* Matches `String` against `Pattern`. For example, `match_pattern("*.pl","foo.pl")` succeeds.
- *The following properties should hold at call time:*

  `Pattern` is a pattern to match against. (patterns:pattern/1)

  `String` is a string (a list of character codes). (basic_props:string/1)

match_pattern/3: PREDICATE

Usage: `match_pattern(Pattern,String,Tail)`

- *Description:* Matches `String` against `Pattern`. `Tail` is the remainder of the string after the match. For example, `match_pattern("??*","foo.pl",Tail)` succeeds, instantiating `Tail` to `"o.pl"`.
- *The following properties should hold at call time:*

  `Pattern` is a pattern to match against. (patterns:pattern/1)

  `String` is a string (a list of character codes). (basic_props:string/1)

  `Tail` is a string (a list of character codes). (basic_props:string/1)

case_insensitive_match/2: PREDICATE

Usage: `case_insensitive_match(Pred1,Pred2)`

– *Description:* Tests if two predicates `Pred1` and `Pred2` match in a case-insensitive way.

letter_match/2: PREDICATE

Usage: `letter_match(X,Y)`

– *Description:* True iff `X` and `Y` represents the same letter

pattern/1: REGTYPE

Special characters for `Pattern` are:

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by a minus sign denotes a range; any character lexically between those two characters, inclusive, is matched. If the first character following the [is a ^ then any character not enclosed is matched. No other character is special inside this construct. To include a] in a character set, you must make it the first character. To include a '-', you must use it in a context where it cannot possibly indicate a range: that is, as the first character, or immediately after a range.
|	Specifies an alternative. Two patterns A and B with | in between form an expression that matches anything that either A or B will match.
{...}	Groups alternatives inside larger patterns.
\	Quotes a special character (including itself).

Usage: `pattern(P)`

– *Description:* `P` is a pattern to match against.

match_pattern_pred/2: PREDICATE

Usage: `match_pattern_pred(Pred1,Pred2)`

– *Description:* Tests if two predicates `Pred1` and `Pred2` match using regular expressions.

154 Graphs

Author(s): Francisco Bueno.

This module implements utilities for work with graphs

154.1 Usage and interface (graphs)

- **Library usage:**
  `:- use_module(library(graphs)).`
- **Exports:**
  - *Predicates:*
    `dgraph_to_ugraph/2`, `dlgraph_to_lgraph/2`, `edges_to_ugraph/2`, `edges_to_lgraph/2`.
  - *Regular Types:*
    `dgraph/1`, `dlgraph/1`.
- **Other modules used:**
  - *System library modules:*
    `sort`, `graphs/ugraphs`, `graphs/lgraphs`.

154.2 Documentation on exports (graphs)

dgraph/1: REGTYPE

`dgraph(Graph)`

A directed graph is a term `graph(V,E)` where `V` is a list of vertices and `E` is a list of edges (none necessarily sorted). Edges are pairs of vertices which are directed, i.e., `(a,b)` represents `a->b`. Two vertices `a` and `b` are equal only if `a==b`.

Usage: `dgraph(Graph)`

- *Description:* `Graph` is a directed graph.

dlgraph/1: REGTYPE

`dlgraph(Graph)`

A labeled directed graph is a directed graph where edges are triples of the form `(a,l,b)` where `l` is the label of the edge `(a,b)`.

Usage: `dlgraph(Graph)`

- *Description:* `Graph` is a directed labeled graph.

dgraph_to_ugraph/2: PREDICATE

Usage: `dgraph_to_ugraph(Graph,UGraph)`

- *Description:* Converts `Graph` to `UGraph`.

– *The following properties should hold at call time:*
`Graph` is currently a term which is not a free variable. (term_typing:nonvar/1)
`UGraph` is a free variable. (term_typing:var/1)
`Graph` is a directed graph. (graphs:dgraph/1)
`UGraph` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Graph` is a directed graph. (graphs:dgraph/1)
`UGraph` is an ugraph. (ugraphs:ugraph/1)

dlgraph_to_lgraph/2: PREDICATE

Usage: `dlgraph_to_lgraph(Graph,LGraph)`

– *Description:* Converts `Edges` to `LGraph`.
– *The following properties should hold at call time:*
`Graph` is currently a term which is not a free variable. (term_typing:nonvar/1)
`LGraph` is a free variable. (term_typing:var/1)
`Graph` is a directed labeled graph. (graphs:dlgraph/1)
`LGraph` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Graph` is a directed labeled graph. (graphs:dlgraph/1)
`LGraph` is a labeled graph of `term` terms. (lgraphs:lgraph/2)

edges_to_ugraph/2: PREDICATE

Usage: `edges_to_ugraph(Edges,UGraph)`

– *Description:* Converts `Graph` to `UGraph`.
– *The following properties should hold at call time:*
`Edges` is currently a term which is not a free variable. (term_typing:nonvar/1)
`UGraph` is a free variable. (term_typing:var/1)
`Edges` is a list of `pair`s. (basic_props:list/2)
`UGraph` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Edges` is a list of `pair`s. (basic_props:list/2)
`UGraph` is an ugraph. (ugraphs:ugraph/1)

edges_to_lgraph/2: PREDICATE

Usage: `edges_to_lgraph(Edges,LGraph)`

– *Description:* Converts `Edges` to `LGraph`.
– *The following properties should hold at call time:*
`Edges` is currently a term which is not a free variable. (term_typing:nonvar/1)
`LGraph` is a free variable. (term_typing:var/1)
`Edges` is a list of `triple`s. (basic_props:list/2)
`LGraph` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Edges` is a list of `triple`s. (basic_props:list/2)
`LGraph` is a labeled graph of `term` terms. (lgraphs:lgraph/2)

154.3 Documentation on internals (graphs)

pair/1: REGTYPE

Usage: `pair(P)`

– *Description:* `P` is a pair `(_,_)`.

triple/1: REGTYPE

Usage: `triple(P)`

– *Description:* `P` is a triple `(_,_,_)`.

155 Unweighted graph-processing utilities

Author(s): Richard A. O'Keefe (original shared code), Mats Carlsson (adapted from original code), Francisco Bueno (modifications), Manuel Carro (modifications).

An unweighted directed graph (ugraph) is represented as a list of (vertex-neighbors) pairs, where the pairs are in standard order (as produced by keysort with unique keys) and the neighbors of each vertex are also in standard order (as produced by sort), and every neighbor appears as a vertex even if it has no neighbors itself.

An undirected graph is represented as a directed graph where for each edge `(U,V)` there is a symmetric edge `(V,U)`.

An edge `(U,V)` is represented as the term `U-V`.

A vertex can be any term. Two vertices are distinct iff they are not identical (`==/2`).

A path is represented as a list of vertices. No vertex can appear twice in a path.

155.1 Usage and interface (ugraphs)

- **Library usage:**
  `:- use_module(library(ugraphs)).`
- **Exports:**
  - *Predicates:*
    `vertices_edges_to_ugraph/3`, `neighbors/3`, `edges/2`, `del_edges/3`, `add_edges/3`, `vertices/2`, `del_vertices/3`, `add_vertices/3`, `transpose/2`, `rooted_subgraph/3`, `point_to/3`.
  - *Regular Types:*
    `ugraph/1`.
- **Other modules used:**
  - *System library modules:*
    `sets`, `sort`.

155.2 Documentation on exports (ugraphs)

vertices_edges_to_ugraph/3: PREDICATE

No further documentation available for this predicate.

neighbors/3: PREDICATE

Usage: `neighbors(Vertex,Graph,Neighbors)`

- *Description:* Is true if `Vertex` is a vertex in `Graph` and `Neighbors` are its neighbors.
- *The following properties should hold at call time:*

`Vertex` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Graph` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Neighbors` is a free variable. (term_typing:var/1)

edges/2: PREDICATE

Usage: `edges(Graph,Edges)`

- *Description:* Unifies `Edges` with the edges in `Graph`.
- *The following properties should hold at call time:*

  `Graph` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Edges` is a free variable. (term_typing:var/1)

del_edges/3: PREDICATE

Usage: `del_edges(Graph1,Edges,Graph2)`

- *Description:* Is true if `Graph2` is `Graph1` with `Edges` removed from it.
- *The following properties should hold at call time:*

  `Graph1` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Edges` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Graph2` is a free variable. (term_typing:var/1)

add_edges/3: PREDICATE

Usage: `add_edges(Graph1,Edges,Graph2)`

- *Description:* Is true if `Graph2` is `Graph1` with `Edges` and their 'to' and 'from' vertices added to it.
- *The following properties should hold at call time:*

  `Graph1` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Edges` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Graph2` is a free variable. (term_typing:var/1)

vertices/2: PREDICATE

Usage: `vertices(Graph,Vertices)`

- *Description:* Unifies `Vertices` with the vertices in `Graph`.
- *The following properties should hold at call time:*

  `Graph` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Vertices` is a free variable. (term_typing:var/1)

del_vertices/3: PREDICATE

Usage: `del_vertices(Graph1,Vertices,Graph2)`

- *Description:* Is true if `Graph2` is `Graph1` with `Vertices` and all edges to and from `Vertices` removed from it.
- *The following properties should hold at call time:*

  `Graph1` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Vertices` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Graph2` is a free variable. (term_typing:var/1)

add_vertices/3: PREDICATE

Usage: `add_vertices(Graph1,Vertices,Graph2)`

- *Description:* Is true if `Graph2` is `Graph1` with `Vertices` added to it.
- *The following properties should hold at call time:*

  `Graph1` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Vertices` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Graph2` is a free variable. (term_typing:var/1)

transpose/2: PREDICATE

Usage: `transpose(Graph,Transpose)`

- *Description:* Is true if `Transpose` is the graph computed by replacing each edge `(u,v)` in `Graph` by its symmetric edge `(v,u)`. It can only be used one way around. The cost is O(N^2).
- *The following properties should hold at call time:*

  `Graph` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Transpose` is a free variable. (term_typing:var/1)

rooted_subgraph/3: PREDICATE

Usage: `rooted_subgraph(Graph,Sources,SubGraph)`

- *Description:* `SubGraph` is the subgraph of `Graph` which is reachable from `Sources`.
- *The following properties should hold at call time:*

  `Graph` is an ugraph. (ugraphs:ugraph/1)

  `Sources` is a list. (basic_props:list/1)

  `SubGraph` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*

  `SubGraph` is an ugraph. (ugraphs:ugraph/1)

point_to/3: PREDICATE

Usage: `point_to(Vertex,Graph,Point_to)`

- *Description:* Is true if `Point_to` is the list of nodes which go directly to `Vertex` in `Graph`.
- *The following properties should hold at call time:*

  `Vertex` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Graph` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Point_to` is a free variable. (term_typing:var/1)

ugraph/1: REGTYPE

Usage: `ugraph(Graph)`

- *Description:* `Graph` is an ugraph.

156 wgraphs (library)

156.1 Usage and interface (wgraphs)

- **Library usage:**
  `:- use_module(library(wgraphs)).`
- **Exports:**
  - *Predicates:*
    `vertices_edges_to_wgraph/3`.
- **Other modules used:**
  - *System library modules:*
    `sets`, `sort`.

156.2 Documentation on exports (wgraphs)

vertices_edges_to_wgraph/3: PREDICATE

No further documentation available for this predicate.

157 Labeled graph-processing utilities

Author(s): Francisco Bueno.

See the comments for the `ugraphs` library.

157.1 Usage and interface (`lgraphs`)

- **Library usage:**
  `:- use_module(library(lgraphs)).`
- **Exports:**
  - *Predicates:*
    `vertices_edges_to_lgraph/3`.
  - *Regular Types:*
    `lgraph/2`.
- **Other modules used:**
  - *System library modules:*
    `sort`, `sets`.

157.2 Documentation on exports (`lgraphs`)

lgraph/2: REGTYPE

Usage: `lgraph(Graph,Type)`

- *Description:* `Graph` is a labeled graph of `Type` terms.

vertices_edges_to_lgraph/3: PREDICATE

`vertices_edges_to_lgraph(Vertices0,Edges,Graph)`

This one is a copy of the same procedure in library(wgraphs) except for the definition of min/3 (ah! - the polimorphism!).

It would only be needed if there are multi-edges, i.e., several edges between the same two vertices.

158 queues (library)

158.1 Usage and interface (queues)

- **Library usage:**
  `:- use_module(library(queues)).`
- **Exports:**
  - *Predicates:*
    `q_empty/1`, `q_insert/3`, `q_member/2`, `q_delete/3`.

158.2 Documentation on exports (queues)

q_empty/1: PREDICATE
No further documentation available for this predicate.

q_insert/3: PREDICATE
No further documentation available for this predicate.

q_member/2: PREDICATE
No further documentation available for this predicate.

q_delete/3: PREDICATE
No further documentation available for this predicate.

159 Random numbers

Author(s): Daniel Cabeza.
This module provides predicates for generating pseudo-random numbers

159.1 Usage and interface (`random`)

- **Library usage:**
  `:- use_module(library(random)).`
- **Exports:**
  - *Predicates:*
    `random/1`, `random/3`, `srandom/1`.
- **Other modules used:**
  - *System library modules:*
    `foreign_interface/foreign_interface_properties`.

159.2 Documentation on exports (`random`)

random/1: PREDICATE

`random(Number)`

`Number` is a (pseudo-) random number in the range [0.0,1.0]

Usage:

- *The following properties should hold at call time:*
  `Number` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*
  `Number` is currently instantiated to a float. (term_typing:float/1)
- *The following properties hold globally:*
  The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

random/3: PREDICATE

`random(Low,Up,Number)`

`Number` is a (pseudo-) random number in the range [`Low`, `Up`]

Usage 1:

- *Description:* If `Low` and `Up` are integers, `Number` is an integer.
- *The following properties should hold at call time:*
  `Low` is an integer. (basic_props:int/1)
  `Up` is an integer. (basic_props:int/1)
  `Number` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*
  `Number` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

Usage 2:

– *The following properties should hold at call time:*

`Low` is a float. (basic_props:flt/1)

`Up` is a number. (basic_props:num/1)

`Number` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`Number` is a float. (basic_props:flt/1)

Usage 3:

– *The following properties should hold at call time:*

`Low` is an integer. (basic_props:int/1)

`Up` is a float. (basic_props:flt/1)

`Number` is a free variable. (term_typing:var/1)

– *The following properties hold upon exit:*

`Number` is a float. (basic_props:flt/1)

srandom/1: PREDICATE

`srandom(Seed)`

Changes the sequence of pseudo-random numbers according to `Seed`. The stating sequence of numbers generated can be duplicated by calling the predicate with `Seed` unbound (the sequence depends on the OS).

Usage:

– *Calls should, and exit will be compatible with:*

`Seed` is an integer. (basic_props:int/1)

– *The following properties hold upon exit:*

`Seed` is an integer. (basic_props:int/1)

– *The following properties hold globally:*

The Prolog predicate `PrologName` is implemented using the function `ForeignName`. The same considerations as above example are to be applied. (foreign_interface_properties:foreign_low/2)

160 Set Operations

Author(s): Lena Flood.
This module implements set operations. Sets are just ordered lists.

160.1 Usage and interface (sets)

- **Library usage:**
  `:- use_module(library(sets)).`
- **Exports:**
  - *Predicates:*
    `insert/3`, `ord_delete/3`, `ord_member/2`, `ord_test_member/3`, `ord_subtract/3`, `ord_intersection/3`, `ord_intersection_diff/4`, `ord_intersect/2`, `ord_subset/2`, `ord_subset_diff/3`, `ord_union/3`, `ord_union_diff/4`, `ord_union_symdiff/4`, `ord_union_change/3`, `merge/3`, `ord_disjoint/2`, `setproduct/3`.
- **Other modules used:**
  - *System library modules:*
    `sort`.

160.2 Documentation on exports (sets)

insert/3: PREDICATE

Usage: `insert(Set1,Element,Set2)`

- *Description:* It is true when `Set2` is `Set1` with `Element` inserted in it, preserving the order.
- *The following properties should hold at call time:*
  `Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Element` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Set2` is a free variable. (term_typing:var/1)

ord_delete/3: PREDICATE

Usage: `ord_delete(Set0,X,Set)`

- *Description:* It succeeds if `Set` is `Set0` without element `X`.
- *The following properties should hold at call time:*
  `Set0` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `X` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Set` is a free variable. (term_typing:var/1)

ord_member/2: PREDICATE

Usage: `ord_member(X,Set)`

- *Description:* It succeeds if `X` is member of `Set`.

– *The following properties should hold at call time:*
`X` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set` is currently a term which is not a free variable. (term_typing:nonvar/1)

ord_test_member/3: PREDICATE

Usage: `ord_test_member(Set,X,Result)`

– *Description:* If `X` is member of `Set` then `Result=yes`. Otherwise `Result=no`.
– *The following properties should hold at call time:*
`Set` is currently a term which is not a free variable. (term_typing:nonvar/1)
`X` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Result` is a free variable. (term_typing:var/1)

ord_subtract/3: PREDICATE

Usage: `ord_subtract(Set1,Set2,Difference)`

– *Description:* It is true when `Difference` contains all and only the elements of `Set1` which are not also in `Set2`.
– *The following properties should hold at call time:*
`Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)

ord_intersection/3: PREDICATE

Usage: `ord_intersection(Set1,Set2,Intersection)`

– *Description:* It is true when `Intersection` is the ordered representation of `Set1` and `Set2`, provided that `Set1` and `Set2` are ordered lists.
– *The following properties should hold at call time:*
`Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)

ord_intersection_diff/4: PREDICATE

Usage: `ord_intersection_diff(Set1,Set2,Intersect,NotIntersect)`

– *Description:* `Intersect` contains those elements which are both in `Set1` and `Set2`, and `NotIntersect` those which are in `Set1` but not in `Set2`.
– *The following properties should hold at call time:*
`Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Intersect` is a free variable. (term_typing:var/1)
`NotIntersect` is a free variable. (term_typing:var/1)

ord_intersect/2: PREDICATE

Usage: `ord_intersect(Xs,Ys)`

- *Description:* Succeeds when the two ordered lists have at least one element in common.
- *The following properties should hold at call time:*

  `Xs` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Ys` is currently a term which is not a free variable. (term_typing:nonvar/1)

ord_subset/2: PREDICATE

Usage: `ord_subset(Xs,Ys)`

- *Description:* Succeeds when every element of `Xs` appears in `Ys`.
- *The following properties should hold at call time:*

  `Xs` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Ys` is currently a term which is not a free variable. (term_typing:nonvar/1)

ord_subset_diff/3: PREDICATE

Usage: `ord_subset_diff(Set1,Set2,Difference)`

- *Description:* It succeeds when every element of `Set1` appears in `Set2` and `Difference` has the elements of `Set2` which are not in `Set1`.
- *The following properties should hold at call time:*

  `Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Difference` is a free variable. (term_typing:var/1)

ord_union/3: PREDICATE

Usage: `ord_union(Set1,Set2,Union)`

- *Description:* It is true when `Union` is the union of `Set1` and `Set2`. When some element occurs in both sets, `Union` retains only one copy.
- *The following properties should hold at call time:*

  `Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)

ord_union_diff/4: PREDICATE

Usage: `ord_union_diff(Set1,Set2,Union,Difference)`

- *Description:* It succeeds when `Union` is the union of `Set1` and `Set2`, and `Difference` is `Set2` set-minus `Set1`.
- *The following properties should hold at call time:*

  `Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Union` is a free variable. (term_typing:var/1)

  `Difference` is a free variable. (term_typing:var/1)

ord_union_symdiff/4: PREDICATE

Usage: `ord_union_symdiff(Set1,Set2,Union,Diff)`

- *Description:* It is true when Diff is the symmetric difference of Set1 and Set2, and Union is the union of Set1 and Set2.
- *The following properties should hold at call time:*

`Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Union` is a free variable. (term_typing:var/1)
`Diff` is a free variable. (term_typing:var/1)

ord_union_change/3: PREDICATE

Usage: `ord_union_change(Set1,Set2,Union)`

- *Description:* `Union` is the union of `Set1` and `Set2` and `Union` is different from `Set2`.
- *The following properties should hold at call time:*

`Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Union` is a free variable. (term_typing:var/1)

merge/3: PREDICATE

Usage: `merge(Set1,Set2,Union)`

- *Description:* See `ord_union/3`.
- *The following properties should hold at call time:*

`Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)

ord_disjoint/2: PREDICATE

Usage: `ord_disjoint(Set1,Set2)`

- *Description:* `Set1` and `Set2` have no element in common.
- *The following properties should hold at call time:*

`Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)

setproduct/3: PREDICATE

Usage: `setproduct(Set1,Set2,Product)`

- *Description:* `Product` has all two element sets such that one element is in `Set1` and the other in `set2`, except that if the same element belongs to both, then the corresponding one element set is in `Product`.
- *The following properties should hold at call time:*

`Set1` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Set2` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Product` is a free variable. (term_typing:var/1)

161 Variable name dictionaries

Author(s): Francisco Bueno, Edison Mera.

161.1 Usage and interface (vndict)

- **Library usage:**
  `:- use_module(library(vndict)).`
- **Exports:**
  - *Predicates:*
    `create_dict/2`, `create_pretty_dict/2`, `complete_dict/3`, `complete_vars_dict/3`, `prune_dict/3`, `sort_dict/2`, `dict2varnamesl/2`, `varnamesl2dict/2`, `find_name/4`, `prettyvars/2`, `rename/2`, `vars_names_dict/3`.
  - *Regular Types:*
    `null_dict/1`, `varnamedict/1`.
- **Other modules used:**
  - *System library modules:*
    `varnames/dict_types`, `idlists`, `terms_vars`, `sets`, `sort`.

161.2 Documentation on exports (vndict)

null_dict/1: REGTYPE

Usage: `null_dict(D)`

- *Description:* `D` is an empty dictionary.

create_dict/2: PREDICATE

Usage: `create_dict(Term,Dict)`

- *Description:* `Dict` has names for all variables in `Term`.
- *The following properties should hold at call time:*
  `Term` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*
  `Dict` is a dictionary of variable names. (vndict:varnamedict/1)

create_pretty_dict/2: PREDICATE

Usage: `create_pretty_dict(Term,Dict)`

- *Description:* `Dict` has names for all variables in `Term`. The difference with `create_dict/2` is that prettier names are generated
- *The following properties should hold at call time:*
  `Term` is any term. (basic_props:term/1)
- *The following properties should hold upon exit:*
  `Dict` is a dictionary of variable names. (vndict:varnamedict/1)

complete_dict/3: PREDICATE

Usage: `complete_dict(Dict,Term,NewDict)`

- *Description:* `NewDict` is `Dict` augmented with the variables of `Term` not yet in `Dict`.
- *The following properties should hold at call time:*

  `Dict` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `NewDict` is a free variable. (term_typing:var/1)

complete_vars_dict/3: PREDICATE

Usage: `complete_vars_dict(Dict,Vars,NewDict)`

- *Description:* `NewDict` is `Dict` augmented with the variables of the list `Vars` not yet in `Dict`.
- *The following properties should hold at call time:*

  `Dict` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Vars` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `NewDict` is a free variable. (term_typing:var/1)

prune_dict/3: PREDICATE

Usage: `prune_dict(Term,Dict,NewDict)`

- *Description:* `NewDict` is `Dict` reduced to just the variables of `Term`.
- *The following properties should hold at call time:*

  `Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Dict` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `NewDict` is a free variable. (term_typing:var/1)

sort_dict/2: PREDICATE

Usage: `sort_dict(D,Dict)`

- *Description:* `D` is sorted into `Dict`.
- *The following properties should hold at call time:*

  `D` is a dictionary of variable names. (vndict:varnamedict/1)
- *The following properties should hold upon exit:*

  `Dict` is a dictionary of variable names. (vndict:varnamedict/1)

dict2varnamesl/2: PREDICATE

Usage: `dict2varnamesl(Dict,VNs)`

- *Description:* Translates `Dict` to `VNs`.
- *The following properties should hold at call time:*

  `Dict` is a dictionary of variable names. (vndict:varnamedict/1)
- *The following properties should hold upon exit:*

  `VNs` is a list of `Name=Var` pairs, where `Var` is a variable and `Name` its name. (dict_types:varnamesl/1)

varnamesl2dict/2: PREDICATE

Usage: `varnamesl2dict(VNs,Dict)`

- *Description:* Translates `VNs` to `Dict`.
- *The following properties should hold at call time:*

  `VNs` is a list of `Name=Var` pairs, where `Var` is a variable and `Name` its name. (dict_types:varnamesl/1)
- *The following properties should hold upon exit:*

  `Dict` is a dictionary of variable names. (vndict:varnamedict/1)

find_name/4: PREDICATE

`find_name(Vars,Names,V,Name)`

Given that `vars_names_dict(Dict,Vars,Names)` holds, it acts as `rename(X,Dict)`, but the name of `X` is given as `Name` instead of unified with it.

prettyvars/2: PREDICATE

Usage:

- *Description:* Give names to the variables in the term `Term` using the dictionary `Dict`. Intended to replace `prettyvars/1` in those places where is possible to get the dictionary of variables.
- *Call and exit should be compatible with:*

  `Arg1` is any term. (basic_props:term/1)

  `Arg2` is a list of `Name=Var` pairs, where `Var` is a variable and `Name` its name. (dict_types:varnamesl/1)
- *The following properties should hold at call time:*

  `Arg2` is currently a term which is not a free variable. (term_typing:nonvar/1)

rename/2: PREDICATE

Usage: `rename(Term,Dict)`

- *Description:* Unifies each variable in `Term` with its name in `Dict`. If no name is found, a new name is created.
- *The following properties should hold at call time:*

  `Dict` is a dictionary of variable names. (vndict:varnamedict/1)

varnamedict/1: REGTYPE

Usage: `varnamedict(D)`

- *Description:* `D` is a dictionary of variable names.

vars_names_dict/3: PREDICATE

Usage: `vars_names_dict(Dict,Vars,Names)`

- *Description:* `Varss` is a sorted list of variables, and `Names` is a list of their names, which correspond in the same order.

– *Call and exit should be compatible with:*
`Dict` is a dictionary of variable names. (vndict:varnamedict/1)
`Vars` is a list. (basic_props:list/1)
`Names` is a list. (basic_props:list/1)

PART X - Miscellaneous standalone utilities

This is the documentation for a set of miscellaneous standalone utilities contained in the `etc` directory of the Ciao distribution.

162 A Program to Help Cleaning your Directories

Author(s): Manuel Carro.

A simple program for traversing a directory tree and deciding which files may be deleted in order to save space and not to loose information.

162.1 Usage (cleandirs)

```
cleandirs [--silent] <initial_dir> <delete_options> <backup_options>
cleandirs explores <initial_dir> (which should be an absolute path)
and looks for backup files and files which can be generated from other
files, using a plausible heuristic aimed at retaining the same amount
of information while recovering some disk space.  The heuristic is
based on the extension of the filename.

Delete options is one of:
   --list: just list the files/directories which are amenable to be deleted,
           but do not delete them. SAFE.
    --ask: list the files/directories and ask for deletion. UNSAFE if you
           make a mistake.
 --delete: just delete the files/directories without asking.  I envy your
           brave soul if you choose this option.

Backup options is one of:
 --includebackups: include backup files in the list of files to check.
 --excludebackups: do not include backup files in the list of files
                   to check.
    --onlybackups: include only backup files in the list of files to check.

Symbolic links are not traversed.  Special files are not checked.
```

Invoking the program with no arguments will return an up-to-date information on the options.

162.2 Known bugs and planned improvements (`cleandirs`)

- Recursive removal of subdirectories relies on the existence of a recursive /bin/rm command in your system.

163 Printing the declarations and code in a file

Author(s): Manuel Hermenegildo.

A simple program for printing assertion information (predicate declarations, property declarations, type declarations, etc.) and printing code-related information (imports, exports, libraries used, etc.) on a file. The file should be a single Ciao or Prolog source file. It uses the Ciao compiler's pass one to do it. This program is specially useful for example for checking what assertions the assertion normalizer is producing from the original assertions in the file or to check what the compiler is actually seeing after some of the syntactic expansions (but before goal translations).

163.1 Usage (fileinfo)

```
fileinfo -asr <filename.asr>
   : pretty prints the contents of <filename.asr>

fileinfo [-v] [-m] <-a|-f|-c|-e> <filename> [libdir1] ... [libdirN]
-v : verbose output (e.g., lists all files read)
-m : restrict info to current module
-a : print assertions
-f : print code and interface (imports/exports, etc.)
-c : print code only
-e : print only errors - useful to check syntax of assertions in file

fileinfo -h
   : print this information

Note that system lib paths *must* be given explicitly, e.g. :

fileinfo -m -c foo.pl \
         /home/clip/System/ciao/lib \
         /home/clip/System/ciao/library \
```

163.2 More detailed explanation of options (fileinfo)

- If the `-a` option is selected, `fileinfo` prints the assertions (only code-oriented assertions – not comment-oriented assertions) in the file *after normalization.* If the `-f` option is selected `fileinfo` prints the file interface, the declarations contained in the file, and the actual code. The `-c` option prints only the code. If the `-e` option is selected `fileinfo` prints only any sintactic and import-export errors found in the file, including the assertions.
- `filename` must be the name of a Prolog or Ciao source file.
- This filename can be followed by other arguments which will be taken to be library directory paths in which to look for files used by the file being analyzed.
- If the `-m` option is selected, only the information related to the current module is printed.
- The `-v` option produces verbose output. This is very useful for debugging, since all the files accessed during assertion normalization are listed.
- In the `-asr` usage, `fileinfo` can be used to print the contents of a `.asr` file in human-readable form.

164 Printing the contents of a bytecode file

Author(s): Daniel Cabeza.

This simple program takes as an argument a bytecode (.po) file and prints out in symbolic form the information contained in the file. It uses compiler and engine builtins to do so, so that it keeps track with changes in bytecode format.

164.1 Usage (viewpo)

```
viewpo <file1>.po
   : print .po contents in symbolic form

viewpo -h
   : print this information
```

165 callgraph (library)

165.1 Usage and interface (callgraph)

- **Library usage:**
  `:- use_module(library(callgraph)).`
- **Exports:**
  - *Predicates:*
    `call_graph/2`, `reachability/4`.
- **Other modules used:**
  - *System library modules:*
    `assertions/c_itf_props`, `sets`, `terms`, `graphs/ugraphs`, `xrefs/xrefsread`.

165.2 Documentation on exports (callgraph)

call_graph/2: PREDICATE

Usage: `call_graph(File,Graph)`

- *Description:* `Graph` is the call-graph of the code in `File`.
- *The following properties should hold at call time:*
  `File` is an atom describing the name of a file. (c_itf_props:filename/1)
  `Graph` is a free variable. (term_typing:var/1)
- *The following properties hold upon exit:*
  `Graph` is an ugraph. (ugraphs:ugraph/1)

reachability/4: PREDICATE

Usage: `reachability(Graph,Sources,Reached,UnReached)`

- *Description:* `Reached` are the vertices in `Graph` reachable from `Sources`, `UnReached` are the rest.
- *The following properties should hold at call time:*
  `Graph` is an ugraph. (ugraphs:ugraph/1)
  `Sources` is a list. (basic_props:list/1)
  `Reached` is a free variable. (term_typing:var/1)
  `UnReached` is a free variable. (term_typing:var/1)

166 Gathering the dependent files for a file

Author(s): Daniel Cabeza, Manuel Hermenegildo.

This simple program takes a single Ciao or Prolog source filename (which is typically the main file of an application). It prints out the list of all the dependent files, i.e., all files needed in order to build the application, including those which reside in libraries. This is particularly useful in Makefiles, for building standalone distributions (e.g., .tar files) automatically.

The filename should be followed by other arguments which will be taken to be library directory paths in which to look for files used by the file being analyzed.

166.1 Usage (get_deps)

```
get_deps [-u <filename>] <filename> [lib_dir1] ... [lib_dirN]
           : return dependent files for <filename>
             found in [lib_dir1] ... [lib_dirN]

get_deps -h
           : print this information
```

167 Finding differences between two Prolog files

Author(s): Francisco Bueno.

This simple program works like the good old diff but for files that contain Prolog code. It prints out the clauses that it finds are different in the files. Its use avoids textual differences such as different variable names and different formatting of the code in the files.

167.1 Usage (pldiff)

```
pldiff <file1> <file2>
   : find differences

pldiff -h
   : print this information
```

but you can also use the program as a library and invoke the predicate:

```
pldiff( <filename> , <filename> )
```

167.2 Known bugs and planned improvements (pldiff)

- Currently uses variant/2 to compare clauses. This is useful, but there should be an option to select the way clauses are compared, e.g., some form of equivalence defined by the user.

168 The Ciao lpmake scripting facility

Author(s): Manuel Hermenegildo, The CLIP Group.

Note: `lpmake` and the `make` library are still under development, and they may change in future releases.

`lpmake` is a Ciao application which uses the Ciao `make` library to implement a dependency-driven scripts in a similar way to the Un*x `make` facility.

The original purpose of the Un*x `make` utility is to determine automatically which pieces of a large program needed to be recompiled, and issue the commands to recompile them. In practice, `make` is often used for many other purposes: it can be used to describe any task where some files must be updated automatically from others whenever these change. `lpmake` can be used for the same types of applications as `make`, and also for some new ones, and, while being simpler, it offers a number of advantages over `make`. The first one is *portability*. When compiled to a bytecode executable `lpmake` runs on any platform where a Ciao engine is available. Also, the fact that typically many of the operations are programmed in Prolog within the makefile, not needing external applications, improves portability further. The second advantage of `lpmake` is *improved programming capabilities*. While `lpmake` is simpler than `make`, `lpmake` allows using the Ciao Prolog language within the scripts. This allows establising more complex dependencies and programming powerful operations within the make file, and without resorting to external packages (e.g., operating system commands), which also helps portability. A final advantage of `lpmake` is that it supports a form of *autodocumentation*: comments associated to targets can be included in the configuration files. Calling `lpmake` in a directory which has such a configuration file explains what commands the configuration file support and what these commands will do.

168.1 General operation

To prepare to use `lpmake`, and in a similar way to `make`, you must write a *configuration file*: a module (typically called `Makefile.pl`) that describes the relationships among files in your program or application, and states the commands for updating each file. In the case of compiling a program, typically the executable file is obtained from object files, which are in turn obtained by compiling source files. Another example is running `latex` and `dvips` on a set of source `.tex` files to generate a document in `dvi` and `postscript` formats.

Once a suitable make file exists, each time you change some source files, simply typing `lpmake` suffices to perform all necessary operations (recompilations, processing text files, etc.). The `lpmake` program uses the dependency rules in the makefile and the last modification times of the files to decide which of the files need to be updated. For each of those files, it issues the commands recorded in the makefile. For example, in the `latex/ dvips` case one rule states that the `.dvi` file whould be updated from the `.tex` files whenever one of them changes and another rule states that the `.ps` file needs to be updated from a `.dvi` file every time it changes. The rules also describe the commands to be issued to update the files.

So, the general process is as follows: `lpmake` executes commands in the configuration file to update one or more target *names*, where *name* is often a program, but can also be a file to be generated or even a "virtual" target. `lpmake` updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist. You can provide command line arguments to `lpmake` to control which files should be regenerated, or how.

168.2 Format of the Configuration File

`lpmake` uses as default configuration file the file `Makefile.pl`, if it is present in the current directory. This can be overridden and another file used by means of the `-m` option. The configuration file must be a *module* and this module must make use of the `make` package. This package

provides syntax for defining the dependency rules and functionality for correctly interpreting these rules.

The configuration file can contain such rules and also arbitrary Ciao Prolog predicates, and can also import other Ciao modules, packages, or make file. This is useful to implement inherintance across diferent configuration files, i.e., the values declared in a configuration file can be easily made to override those defined in another, using the standard Ciao rules for module overriding, reexport, etc. The syntax of the rules is described in Chapter 92 [The Ciao Make Package], page 481, together with some examples.

168.3 lpmake usage

```
Supported command line options:

lpmake [-v] [-d Name1=Value1] ... [-d Namen=Valuen] \
<command1> ... <commandn>

  Process commands <command1> ... <commandn>, using file 'Makefile.pl'
  or directory 'installer' in the current directory as configuration
  file. The configuration file must be a module.

  The optional argument '-v' produces verbose output, reporting on the
  processing of the dependency rules.  Very useful for debugging
  makefiles.

  The argument '-d' indicates that a variable definition Name=Value
  follows.  The effect of this is adding the fact 'name_value(Name, Value).'
  (i.e., 'name_value(Name) := Value.'), defined in the module
  library(make(make_rt)).

lpmake [-v] [-d Name1=Value1] ... [-d Namen=Valuen] \
[[-m|-l] <.../Configfile1.pl>] [[-m|-l] <.../Configfilen.pl>] \
        <command1> ... <commandn>

  Same as above, but using files <.../Configfilex.pl> as configuration
  file. One or more configuration files can be used. When using more
  than one configuration file, the additional configuration files are
  loaded dynamically into the first one with the predicate
  register_config_file/1.  Using -l instead of -m indicates that this
  configuration file is a library module (i.e., it will be looked for
  in the libraries).

lpmake -h     [ [-m|-l] <.../Configfile.pl> ]
lpmake -help  [ [-m|-l] <.../Configfile.pl> ]
lpmake --help [ [-m|-l] <.../Configfile.pl> ]

  Print this help message. If a configuration file is available in the
  current directory or is given as an option, and the commands in it
  are commented, then information on these commands is also printed.
```

168.4 Acknowledgments (`lpmake`)

Some parts of the documentation are taken from the documentation of GNU's `gmake`.

168.5 Known bugs and planned improvements (`lpmake`)

- Rename or add a default directory makedir/Makedir, instead of installer directory, as recommended by Manuel Herme. – EMM

169 Find out which architecture we are running on

Author(s): Manuel Carro, Robert Manchek.

The architecure and operating system the engine is compiled for determines whether we can use or not certain libraries. This script, taken from a PVM distribution, uses a heuristic (which may need to be tuned from time to time) to find out the platform. It returns a string which is used throughout the engine (in #ifdefs) to enable/disable certain characteristics.

169.1 Usage (ciao_get_arch)

```
Usage: ciao_get_arch
```

169.2 More details (ciao_get_arch)

Look at the script itself...

170 Print out WAM code

Author(s): Manuel Carro.

This program prints to standard output a symbolic form of the Wam code the compiler generates for a given source file.

170.1 Usage (compiler_output)

```
Print WAM code for a .pl file

Usage: compiler_output <file.pl>
```

171 Compile Ciao in an arbitrary subdirectory

Author(s): Manuel Carro.

A simple bash script for compile the whole Ciao distribution, in order to test automatically whether something is broken.

171.1 Usage (auto_compile_ciao)

```
Usage: auto_compile_ciao ciaodir_name destdir_name
```

171.2 More details (auto_compile_ciao)

The script copies the ciao distribution passed on as first argument to a subdirectory of the directory name passed as second argument. The subdirectory is always the same, in order to speed up tests made repeteadly in a distribution. It also deletes from the destination files which do not appear any longer in the sources, and tries not to copy files which will eventually be generated or deleted prior to compilation.

After that, the `SETTINGS` file is updated to reflect the new location, the sources are cleaned up, and a regular compilation is started.

172 Gathering all modules in a directory

Author(s): Daniel Cabeza, Manuel Carro.

A simple bash script for collecting files which are modules, in order to compile automatically the CIAO libraries.

172.1 Usage (collect_modules)

```
Usage: collect_modules module_name
```

172.2 More details (collect_modules)

Look at the Makefiles in lib and library

PART XI - Contributed libraries

Author(s): The CLIP Group.

This part includes a number of libraries which have contributed by users of the Ciao system. Over time, some of these libraries are moved to the main library directories of the system.

173 Block Declarations

Author(s): Rémy Haemmerlé.
Version: 0.1 (2008/25/5)
This package provides compatibility with SICStus' block declarations

173.1 Usage and interface (`block_doc`)

- **Library usage:**
  `:- use_package(block).`
  or
  `:- module(...,...,[block]).`
- **New operators defined:**
  `block/1` [1150,fx].
- **New declarations defined:**
  `block/1`.

173.2 Documentation on new declarations (`block_doc`)

block/1: DECLARATION

Usage: `:- block(BlockSpecs).`

- *Description:* In this declaration `BlockSpecs` specifies a disjunction of conditions. Each condition is of the form `predname(C1, ..., CN)` where each `CI` is either a '`-`' if the call must suspend until the corresponding argument is bound, or anything else otherwise.
- *Convention:* The recommended style is to write the block declarations in front of the source code of the predicate they refer to. Indeed, they are part of the source code of the predicate and must precede the first clause. Moreover it is suggested to use '`?`' for specifying non conditioned arguments.
- *Example :* The following definition calls to `merge/3` having uninstantiated arguments in the first and third position or in the second and third position will suspend.

```
:- block merge(-,?,-), merge(?,-,-).

merge([], Y, Y).
merge(X, [], X).
merge([H|X], [E|Y], [H|Z]) :- H @< E,  merge(X, [E|Y], Z).
merge([H|X], [E|Y], [E|Z]) :- H @>= E, merge([H|X], Y, Z).
```

- *The following properties hold at call time:*
  `BlockSpecs` is a sequence or list of `callable`s. (basic_props:sequence_or_list/2)

174 A Chart Library

Author(s): Isabel Martín García.

This library is intended to eaose the task of displaying some graphical results. This library allows the programmer to visualize different graphs and tables without knowing anything about specific graphical packages.

You need to install the BLT package in your computer. BLT is an extension to the Tk toolkit and it does not require any patching of the Tcl or Tk source files. You can find it in `http://www.tcltk.com/blt/index.html`

Basically, when the user invokes a predicate, the library (internally) creates a bltwish interpreter and passes the information through a socket to display the required widget. The interpreter parses the received commands and executes them.

The predicates exported by this library can be classified in four main groups, according to the types of representation they provide.

- bar charts
- line graphs
- scatter graphs
- tables

To represent graphs, the Cartesian coordinate system is used. I have tried to show simple samples for every library exported predicate in order to indicate how to call them.

174.1 Bar charts

In this section we shall introduce the general issues about the set of barchart predicates. By calling the predicates that pertain to this group a bar chart for plotting two-dimensional data (X-Y coordinates) can be created. A bar chart is a graphic means of comparing numbers by displaying bars of lengths proportional to the y-coordinates they represented. The barchart widget has many configurable options such as title, header text, legend and so on. You can configure the appearance of the bars as well. The bar chart widget has the following components:

Header text
: The text displayed at the top of the window. If it is `''` no text will be displayed.

Save button
: The button placed below the header text. Pops up a dialog box for the user to select a file to save the graphic in PostScript format.[1]

Bar chart title
: The title of the graph. It is displayed at the top of the bar chart graph. If text is `''` no title will be displayed.

X axis title
: X axis title. If text is `''` no x axis title will be displayed.

Y axis title
: Y axis title. If text is `''` no y axis title will be displayed.

X axis
: X coordinate axis. The x axis is drawn at the bottom margin of the bar chart graph. The x axis consists of the axis line, ticks and tick labels. Tick labels can be numbers or plain text. If the labels are numbers, they could be displayed at uniform intervals (the numbers are treated as normal text) or depending on its x-coodinate value. You can also set limits (maximum and minimum) for the x axis, but only if the tick labels are numeric.

[1] Limitation: Some printers can have problems if the PostScript file is too complex (i.e. too many points/lines appear in the picture).

Y axis
: Y coordinate axis. You can set limits (maximum and minimum) for the y axis. The y axis is drawn at the right margin of the bar chart graph.The y axis consists of the axis line, ticks and tick labels. The tick labels are numeric values determined from the data and are drawn at uniform intervals.

Bar chart graph
: This is the plotting area, placed in the center of the window and surrounded by the axes, the axis titles and the legend (if any). The range of the axes controls what region of the data is plotted. By default, the minimum and maximum limits are determined from the data, but you can set them (as mentioned before). Data points outside the minimum and maximum value of the axes are not plotted.

Legend
: The legend displays the name and symbol of each bar. The legend is placed in the right margin of the Bar chart graph.

Footer text
: Text displayed at the lower part of the window. If text is '' no header text will be displayed.

Quit button
: Button placed below the footer text. Click it to close the window.

All of them are arranged in a window. However you can, for example, show a bar chart window without legend or header text. Other configuration options will be explained later.

In addition to the window appearance there is another important issue about the bar chart window, namely its behaviour in response to user actions. The association user actions to response is called *bindings*. The main bindings currently specified are the following:

Default bindings
: Those are well known by most users. They are related to the frame displayed around the window. As you know, you can interactively move, resize, close, iconify, deiconify, send to another desktop etc. a window.

Bindings related to bar chart graph and its legend
: Clicking the left mouse key over a legend element, the corresponding bar turns out into red. After clicking again, the bar toggles to its original look. In addition, you can do zoom-in by pressing the left mouse key over the bar chart graph and dragging to select an area. To zoom out simply press the right mouse button.

  When the pointer passes over the plotting area the cross hairs are drawn. The cross hairs consists of two intersecting lines (one vertical and one horizontal). Besides, if the pointer is over a legend element, its background changes.

Bindings related to buttons
: There are two buttons in the main widget. Clicking the mouse on the Save button a "Save as" dialog box is popped up. The user can select a file to save the graph. If the user choose a file that already exists, the dialog box prompts the user for confirmation on whether the existing file should be overwritten or not. Furthermore, you can close the widget by clicking on the Quit button.

  When the pointer passes over a button the button color changes.

The predicates that belong to this group are those whose names begin with **barchart** and **genmultibar**. If you take a look at the predicate names that pertain to this group, you will notice that they are not self-explanatory. It would have been better to name the predicates in a way that allows the user to identify the predicate features by its name, but it would bring about very long names (i.e barchart_WithoutLegend_BarsAtUniformIntervals_RandomBarsColors). For this reason I decided to simply add a number after barchart to name them.

174.2 Line graphs

It is frequently the case that several datasets need to be displayed on the same plot. If so, you may wish to distinguish the points in different datasets by joining them by lines of different color, or by plotting with symbols of different types. This set of predicates allows the programmer to represent two-dimensional data (X-Y coordinates). Each dataset contains x and y vectors containing the coordinates of the data. You can configure the appearance of the points and the lines which the points are connected with. The configurable line graph components are:

line graph
: This is the plotting area, placed in the center of the window and surrounded by the axes, the axes titles and the legend (if any). The range of the axes controls what region of the data is plotted. By default, the minimum and maximum limits are determined from the data, but you can set them. Data points outside the minimum and maximum value of the axes are not plotted. You can specify how connecting line segments joining successive datapoints are drawn by setting the `Smooth` argument. Smooth can be either linear, step, natural and quadratic. Furthermore, you can select the appearance of the points and lines.

Legend
: The legend displays the name and symbol of each line. The legend is placed in the right margin of the graph.

The elements header, footer, quit and save buttons, the titles and the axes are quite similar to those in barchart graphs, except in that the tick labels will be numbers. All of them are arranged in a window by the geometry manager. However you can, as we mentioned in the above paragraphs, show a line graph window without any titles or footer text. Other configuration options will be explained later in this section or in the corresponding modules.

Related to the behaviour of the widgets in response to user actions (bindings) we will remark the following features:

Bindings related to line graph and its legend
: Clicking the left mouse key over a legend element, the corresponding line turns out into blue. Repeating the action reverts the line to its original color. Moreover, you can do zoom-in by clicking the left mouse key over the bar chart graph and dragging a rectangle defining the area you want to zoom in. To zoom out simply press the right mouse button.

  When the pointer passes over the plotting area the cross hairs are drawn. The cross hairs consists of two intersecting lines (one vertical and one horizontal). Besides, if the pointer is over a legend element, its background changes.

Other bindings
: The default bindings and the bindings related to the save and quit buttons are similar to those in the bar chart graphs.

The predicates that belong to this group are those whose names begin with **graph_**.

174.3 Scatter graphs

The challenge of this section is to introduce some general aspects about the scatter graph predicates group. By invoking the scatter graph predicates the user can represent two-dimensional point datasets. Often you need to display one or several point datasets on the same plot. If so, you may wish to distinguish the points that pertain to different datasets by using plotting symbols of different types, or by displaying them in different colors. This set of predicates allows you to represent two-dimensional data (X-Y coordinates). Each dataset contains x and y vectors containing the coordinates of the data. You can configure the appearance of the points. The configurable scatter graph components are:

scatter graph
: This is the plotting area, placed in the center of the window and surrounded by the axes, the axes titles and the legend (if any). The range of the axes controls what region of the data is plotted. By default, the minimum and maximum limits are determined from the data, but you can set them (as we mentioned before). Data points outside the minimum and maximum value of the axes are not plotted. The user can select the appearance of the points.

Legend
: The legend displays the name and symbol of each point dataset. The legend is drawn in the right margin of the graph.

The elements header, footer, quit and save buttons, the titles and the axes are similar to those in barchart graphs except for that, as in line graphs, the tick labels will be numbers. All of them are arranged in a window by the geometry manager. However you can, for example, show a scatter graph window without titles or footer text, as we mentioned before. Other configuration options will be explained later, in the corresponding modules.

Related to the behaviour of the widgets in response to user actions (bindings) the following features are:

Bindings related to scatter graph and its legend
: Clicking the left mouse key over a legend element, the points which belong to the corresponding dataset turn out into blue. Repeating the action toggles the point dataset to its original color. Moreover, you can do zoom-in by clicking the left mouse key over the bar chart graph and dragging a rectangle defining the area you want to zoom-in on. To do zoom-out simply press the right mouse button.

  When the pointer passes over the plotting area the cross hairs are drawn. The cross hairs consists of two intersecting lines (one vertical and one horizontal). Besides, if the pointer is over a legend element, its background changes.

Other bindings
: The default bindings and the bindings related to the save and quit buttons are similar to those in the bar chart graphs.

The predicates that belong to this group are those whose names began with **scattergraph_**.

174.4 Tables

The purpose of this section is to allow the user to display results in a table. A table is a regular structure in which:

- Every row has the same number of columns, or
- Every column has the same number of rows.

The widget configurable components are as follows:

Title
: Title of the widget, it is displayed centered at the top of the canvas. If text is '' no title will be displayed.

Header text
: Left centered text displayed bellow the title. If text is '' no header text will be displayed.

Table
: Is placed in the center of the window. The table is composed by cells ordered in rows and columns. The cell values can be either any kind of text or numbers and they can be empty as well (see the type definition in the corresponding chapter module). A table is a list of lists. Each sublist is a row, so every sublist in the table must contain the same number of alements.

Footer text
: Left centered text displayed at the lower part of the window. If text is '' no header text will be displayed.

Quit button
: Button placed below the footer text. You can click it to close the window.

If the arguments are not in a correct format an exception will be thrown. Moreover, these widgets have the default bindings and the binding related to the quit button:

The set of predicates that belongs to this group are those which names begin with **table_widget**.

174.5 Overview of widgets

Although you don't have to worry about how to arrange the widgets, here is an overview of how Tcl-tk, the underlying graphical system currently used by chartlib, performs this task. Quoting from the book *Tcl and Tk toolkit*, John K. Ousterhout.

The X Window System provides many facilities for manipulating windows in displays. The root window may have any number of child windows, each of wich is called a top-level window. Top-level windows may have children of their own, wich may have also children, and so on. The descendants of top-level windows are called internal windows. Internal windows are usedfor individual controls such as buttons, text entries, and for grouping controls together. An X-application tipically manages several top-level windows. Tk uses X to implement a set of controls with the Motif look and feel. These controls are called widgets. Each widget is implemented using one X window, and the terms "window" and "widget" will be used interchangeably in this document. As with windows, widgets are nested in hierarchical structures. In this library top-level widgets (nonleaf or main) are just containers for organizing and arranging the leaf widgets (components). Thereby, the barchart widget is a top-level window wich contains some widget components.

Probably the most painstaking aspect of building a graphical application is getting the placement and size of the widgets just right. It usually takes many iterations to align widgets and adjust their spacing. That's because managing the geometry of widgets is simply not a packing problem, but also graphical design problem. Attributes such as alignment, symmetry, and balance are more important than minimizing the amount of space used for packing. Tk is similar to other X toolkits in that it does not allow widgets to determine their own geometries. A widget will not even appeared unless it is managed by a geometry manager. This separation of geometry management from internal widget behaviour allows multiple geometry managers to exist simultaneously and permits any widget to be used with any geometry manager. A geometry manager's job is to arrange one or more *slave* widgets relative to a *master* widgets. There are some geometry managers in Tk such as pack, place and canvas widget. We will use another one call table.

The table geometry manager arranges widgets in a table. It's easy to align widgets (horizontally and vertically) or to create empty space to balance the arrangement of the widgets. Widgets (called slaves in the Tk parlance) are arranged inside a containing widget (called the master). Widgets are positioned at row,column locations and may span any number of rows or columns. More than one widget can occupy a single location. The placement of widget windows determines both the size and arrangement of the table. The table queries the requested size of each widget. The requested size of a widget is the natural size of the widget (before the widget is shrunk or expanded). The height of each row and the width of each column is the largest widget spanning that row or column. The size of the table is in turn the sum of the row and column sizes. This is the table's normal size. The total number of rows and columns in a table is determined from the indices specified. The table grows dynamically as windows are added at larger indices.

174.6 Usage and interface (chartlib)

- **Library usage:**
  `:- use_module(library(chartlib)).`
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/genbar2`, `chartlib/genbar3`, `chartlib/genbar4`, `chartlib/genmultibar`, `chartlib/table_widget1`, `chartlib/table_widget2`, `chartlib/table_widget3`, `chartlib/table_widget4`, `chartlib/gengraph1`, `chartlib/gengraph2`, `chartlib/chartlib_errhandle`.

174.7 Documentation on exports (chartlib)

barchart1/7: (UNDOC_REEXPORT)
Imported from `genbar1` (see the corresponding documentation for details).

barchart1/9: (UNDOC_REEXPORT)
Imported from `genbar1` (see the corresponding documentation for details).

percentbarchart1/7: (UNDOC_REEXPORT)
Imported from `genbar1` (see the corresponding documentation for details).

barchart2/7: (UNDOC_REEXPORT)
Imported from `genbar2` (see the corresponding documentation for details).

barchart2/11: (UNDOC_REEXPORT)
Imported from `genbar2` (see the corresponding documentation for details).

percentbarchart2/7: (UNDOC_REEXPORT)
Imported from `genbar2` (see the corresponding documentation for details).

barchart3/7: (UNDOC_REEXPORT)
Imported from `genbar3` (see the corresponding documentation for details).

barchart3/9: (UNDOC_REEXPORT)
Imported from `genbar3` (see the corresponding documentation for details).

percentbarchart3/7: (UNDOC_REEXPORT)
Imported from `genbar3` (see the corresponding documentation for details).

barchart4/7: (UNDOC_REEXPORT)
Imported from `genbar4` (see the corresponding documentation for details).

barchart4/11: (UNDOC_REEXPORT)
Imported from `genbar4` (see the corresponding documentation for details).

percentbarchart4/7: (UNDOC_REEXPORT)
Imported from `genbar4` (see the corresponding documentation for details).

multibarchart/8: (UNDOC_REEXPORT)
Imported from `genmultibar` (see the corresponding documentation for details).

multibarchart/10: (UNDOC_REEXPORT)
Imported from `genmultibar` (see the corresponding documentation for details).

tablewidget1/4: (UNDOC_REEXPORT)
Imported from `table_widget1` (see the corresponding documentation for details).

tablewidget1/5: (UNDOC_REEXPORT)
Imported from `table_widget1` (see the corresponding documentation for details).

tablewidget2/4: (UNDOC_REEXPORT)
Imported from `table_widget2` (see the corresponding documentation for details).

tablewidget2/5: (UNDOC_REEXPORT)
Imported from `table_widget2` (see the corresponding documentation for details).

tablewidget3/4: (UNDOC_REEXPORT)
Imported from `table_widget3` (see the corresponding documentation for details).

tablewidget3/5: (UNDOC_REEXPORT)
Imported from `table_widget3` (see the corresponding documentation for details).

tablewidget4/4: (UNDOC_REEXPORT)
Imported from `table_widget4` (see the corresponding documentation for details).

tablewidget4/5: (UNDOC_REEXPORT)
Imported from `table_widget4` (see the corresponding documentation for details).

graph_b1/9: (UNDOC_REEXPORT)
Imported from `gengraph1` (see the corresponding documentation for details).

graph_b1/13: (UNDOC_REEXPORT)
Imported from `gengraph1` (see the corresponding documentation for details).

graph_w1/9: (UNDOC_REEXPORT)
Imported from `gengraph1` (see the corresponding documentation for details).

graph_w1/13: (UNDOC_REEXPORT)
Imported from `gengraph1` (see the corresponding documentation for details).

scattergraph_b1/8: (UNDOC_REEXPORT)
Imported from `gengraph1` (see the corresponding documentation for details).

scattergraph_b1/12: (UNDOC_REEXPORT)
Imported from `gengraph1` (see the corresponding documentation for details).

scattergraph_w1/8: (UNDOC_REEXPORT)
Imported from `gengraph1` (see the corresponding documentation for details).

scattergraph_w1/12: (UNDOC_REEXPORT)
Imported from `gengraph1` (see the corresponding documentation for details).

graph_b2/9: (UNDOC_REEXPORT)
Imported from `gengraph2` (see the corresponding documentation for details).

graph_b2/13: (UNDOC_REEXPORT)
Imported from `gengraph2` (see the corresponding documentation for details).

graph_w2/9: (UNDOC_REEXPORT)
Imported from `gengraph2` (see the corresponding documentation for details).

graph_w2/13: (UNDOC_REEXPORT)
Imported from `gengraph2` (see the corresponding documentation for details).

scattergraph_b2/8: (UNDOC_REEXPORT)
Imported from `gengraph2` (see the corresponding documentation for details).

scattergraph_b2/12: (UNDOC_REEXPORT)
Imported from `gengraph2` (see the corresponding documentation for details).

scattergraph_w2/8: (UNDOC_REEXPORT)
Imported from `gengraph2` (see the corresponding documentation for details).

scattergraph_w2/12: (UNDOC_REEXPORT)
Imported from `gengraph2` (see the corresponding documentation for details).

chartlib_text_error_protect/1: (UNDOC_REEXPORT)
Imported from `chartlib_errhandle` (see the corresponding documentation for details).

chartlib_visual_error_protect/1: (UNDOC_REEXPORT)
Imported from `chartlib_errhandle` (see the corresponding documentation for details).

174.8 Known bugs and planned improvements (`chartlib`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

175 Low level Interface between Prolog and blt

Author(s): Isabel Martín García.

This module exports some predicates to interact with Tcl-tk, particularly with the bltwish program. Bltwish is a windowing shell consisting of the Tcl command languaje, the Tk toolkit plus the additional commands that comes with the BLT library and a main program that reads commands. It creates a main window and then processes Tcl commands.

175.1 Usage and interface (`bltclass`)

- **Library usage:**
  `:- use_module(library(bltclass)).`
- **Exports:**
  - *Predicates:*
    `new_interp/1`, `tcltk_raw_code/2`, `interp_file/2`.
  - *Regular Types:*
    `bltwish_interp/1`.
- **Other modules used:**
  - *System library modules:*
    `strings`.

175.2 Documentation on exports (`bltclass`)

new_interp/1: PREDICATE

`new_interp(Interp)`

Creates a bltwish interprter and returns a socket. The socket allows the comunication between Prolog and Tcl-tk. Thus, bltwish receives the commands through the socket.

Usage:

- *The following properties should hold at call time:*
  `Interp` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  bltclass:bltwish_interp(Interp) (bltclass:bltwish_interp/1)

tcltk_raw_code/2: PREDICATE

`tcltk_raw_code(Command_Line,Interp)`

Sends a command line to the interpreter. Tcl-tk parses and executes it.

Usage:

- *The following properties should hold at call time:*
  `Command_Line` is a string (a list of character codes). (basic_props:string/1)
  bltclass:bltwish_interp(Interp) (bltclass:bltwish_interp/1)

bltwish_interp/1: REGTYPE

`bltwish_interp(Interp)`

This type defines a bltwish interpreter. In fact, the bltwish interpreter receives the commands through the socket created.

```
bltwish_interp(Interp) :-
        stream(Interp).
```

interp_file/2: PREDICATE

`interp_file(File,Interp)`

Sends the script file (File) to the interpreter through the socket. A script file is a file that contains commands that Tcl-tk can execute.

Usage:

– *The following properties should hold at call time:*

`File` is a source name. (streams_basic:sourcename/1)

bltclass:bltwish_interp(Interp) (bltclass:bltwish_interp/1)

176 Error Handler for Chartlib

Author(s): Isabel Martín García.

This module is an error handler. If the format of the arguments is not correct in a call to a chartlib predicate an exception will be thrown . You can wrap the chartlib predicates with the predicates exported by this module to handle automatically the errors if any.

176.1 Usage and interface (`chartlib_errhandle`)

- **Library usage:**
  `:- use_module(library(chartlib_errhandle)).`
- **Exports:**
  - *Predicates:*
    `chartlib_text_error_protect/1`, `chartlib_visual_error_protect/1`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/bltclass`, `chartlib/install_utils`.

176.2 Documentation on exports (`chartlib_errhandle`)

chartlib_text_error_protect/1: PREDICATE

`chartlib_text_error_protect(G)`

This predicate catches the thrown exception and sends it to the appropiate handler. The handler will show the error message in the standard output.

Meta-predicate with arguments: `chartlib_text_error_protect(goal)`.

Usage:

- *The following properties should hold at call time:*
  `G` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

chartlib_visual_error_protect/1: PREDICATE

`chartlib_visual_error_protect(G)`

This predicate catches the thrown exception and sends it to the appropiate handler. The handler will pop up a message box.

Meta-predicate with arguments: `chartlib_visual_error_protect(goal)`.

Usage:

- *The following properties should hold at call time:*
  `G` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

176.3 Documentation on internals (chartlib_errhandle)

handler_type/1: REGTYPE

`handler_type(X)`

The library chartlib includes two error handlers already programmed.

```
handler_type(text).
handler_type(visual).
```

error_message/2: PREDICATE

`error_message(ErrorCode,ErrorMessage)`

Binds the error code with its corresponding text message.

Usage:

– *The following properties should hold at call time:*

`ErrorCode` is an atom. (basic_props:atm/1)

`ErrorMessage` is an atom. (basic_props:atm/1)

error_file/2: PREDICATE

`error_file(ErrorCode,ErrorFile)`

Binds the error code with its corresponding script error file.

Usage:

– *The following properties should hold at call time:*

`ErrorCode` is an atom. (basic_props:atm/1)

`ErrorFile` is an atom. (basic_props:atm/1)

177 Color and Pattern Library

Author(s): Isabel Martín García.

This module contains predicates to access and check conformance to the available colors and patterns.

177.1 Usage and interface (color_pattern)

- **Library usage:**
  `:- use_module(library(color_pattern)).`
- **Exports:**
  - *Predicates:*
    `color/2`, `pattern/2`, `random_color/1`, `random_lightcolor/1`, `random_darkcolor/1`, `random_pattern/1`.
  - *Regular Types:*
    `color/1`, `pattern/1`.
- **Other modules used:**
  - *System library modules:*
    `random/random`.

177.2 Documentation on exports (color_pattern)

color/1: REGTYPE

```
color(Color)

    color('GreenYellow').
    color('Yellow').
    color('White').
    color('Wheat').
    color('BlueViolet').
    color('Violet').
    color('MediumTurquoise').
    color('DarkTurquoise').
    color('Turquoise').
    color('Thistle').
    color('Tan').
    color('Sienna').
    color('Salmon').
    color('VioletRed').
    color('OrangeRed').
    color('MediumVioletRed').
    color('IndianRed').
    color('Red').
    color('Plum').
    color('Pink').
    color('MediumOrchid').
    color('DarkOrchid').
```

```
color('Orchid').
color('Orange').
color('Maroon').
color('Magenta').
color('Khaki').
color('Grey').
color('LightGray').
color('DimGray').
color('DarkSlateGray').
color('YellowGreen').
color('SpringGreen').
color('SeaGreen').
color('PaleGreen').
color('MediumSpringGreen').
color('MediumSeaGreen').
color('LimeGreen').
color('ForestGreen').
color('DarkOliveGreen').
color('DarkGreen').
color('Green').
color('Goldenrod').
color('Gold').
color('Brown').
color('Firebrick').
color('Cyan').
color('Coral').
color('SteelBlue').
color('SlateBlue').
color('SkyBlue').
color('Navy').
color('MidnightBlue').
color('MediumSlateBlue').
color('MediumBlue').
color('LightSteelBlue').
color('LightBlue').
color('DarkSlateBlue').
color('CornflowerBlue').
color('CadetBlue').
color('Blue').
color('Black').
color('MediumAquamarine').
color('Aquamarine').
```

Defines available colors for elements such as points, lines or bars.

color/2: PREDICATE

Usage: `color(C1,C2)`

– *Description:* Test whether the color `C1` is a valid color or not. If `C1` is a variable the predicate will choose a valid color randomly. If `C1` is a ground term that is not a valid color an exception (error9) will be thrown

– *The following properties should hold at call time:*
color_pattern:color(C1) (color_pattern:color/1)
– *The following properties should hold upon exit:*
color_pattern:color(C2) (color_pattern:color/1)

pattern/1: REGTYPE

```
pattern(Pattern)
```

```
pattern(pattern1).
pattern(pattern2).
pattern(pattern3).
pattern(pattern4).
pattern(pattern5).
pattern(pattern6).
pattern(pattern7).
pattern(pattern8).
pattern(pattern9).
```

Defines valid patterns used in the stipple style bar attribute.

pattern/2: PREDICATE

Usage: `pattern(P1,P2)`

– *Description:* Test whether the pattern `P1` is a valid pattern or not. If `P1` is a variable the predicate will choose a valid pattern randomly. If `P1` is a ground term that is not a valid pattern an exception (error10) will be thrown.
– *The following properties should hold at call time:*
color_pattern:pattern(P1) (color_pattern:pattern/1)
– *The following properties should hold upon exit:*
color_pattern:pattern(P2) (color_pattern:pattern/1)

random_color/1: PREDICATE

```
random_color(Color)
```

This predicate choose a valid color among the availables randomly.

Usage:

– *The following properties should hold at call time:*
color_pattern:color(Color) (color_pattern:color/1)

random_lightcolor/1: PREDICATE

```
random_lightcolor(Color)
```

This predicate choose a valid light color among the availables randomly.

Usage:

– *The following properties should hold at call time:*
`Color` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
color_pattern:color(Color) (color_pattern:color/1)

random_darkcolor/1: PREDICATE

`random_darkcolor(Color)`

This predicate choose a valid dark color among the availables randomly.

Usage:

- *The following properties should hold at call time:*
  `Color` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  color_pattern:color(Color) (color_pattern:color/1)

random_pattern/1: PREDICATE

`random_pattern(Pattern)`

This predicate choose a valid pattern among the availables randomly.

Usage:

- *The following properties should hold at call time:*
  `Pattern` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  color_pattern:pattern(Pattern) (color_pattern:pattern/1)

178 Barchart widgets - 1

Author(s): Isabel Martín García.

This module defines predicates to show barchart widgets. The three predicates exported by this module plot two-variable data as regular bars in a window. They all share the following features:

- No numeric values for the `x` axis are needed because they will be interpreted as labels. See `xbarelement1/1` definition type.
- The bars will be displayed at uniform intervals.
- The user can either select the appearance of the bars (background color, foreground color and stipple style) or not. See the `xbarelement1` type definition. Thus, the user can call each predicate in two ways.
- The bar chart has a legend. One entry (symbol and label) per bar.
- If you don't want to display text in the elements header, barchart title, x axis title, y axis title or footer, simply type `''` as the value of the argument.
- The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception `error2` will be thrown. If the vectors contains elements but are not correct, the exception `error1` or `error3` will be thrown, depending on the error type. `error1` means that `XVector` and `YVector` do not contain the same number of elements and `error3` indicates that not all the `XVector` elements contain a correct number of attributes .

Particular features will be pointed out in the corresponding predicate.

178.1 Usage and interface (genbar1)

- **Library usage:**
  `:- use_module(library(genbar1)).`
- **Exports:**
  - *Predicates:*
    `barchart1/7`, `barchart1/9`, `percentbarchart1/7`.
  - *Regular Types:*
    `yelement/1`, `axis_limit/1`, `header/1`, `title/1`, `footer/1`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/bltclass`, `chartlib/test_format`, `chartlib/color_pattern`, `chartlib/install_utils`, `lists`, `random/random`.

178.2 Documentation on exports (genbar1)

barchart1/7: PREDICATE

`barchart1(Header,BarchartTitle,XTitle,XVector,YTitle,YVector,Footer)`

The `y` axis range is determined from the limits of the data. Two examples are given to demonstrate clearly how to call the predicates. In the first example the user sets the bar appearance, in the second one the appearance features will be chosen by the system and

the colors that have been assigned to the variables Color1, Color2 and Pattern will be shown also.

Example 1:

```
barchart1('This is the header text',
  'Barchart title',
  'xaxistitle',
  [ ['bar1','legend_element1','Blue','Yellow','pattern1'],
          ['bar2','legend_element2','Plum','SeaGreen','pattern2'],
          ['bar3','legend_element3','Turquoise','Yellow',
    'pattern5'] ],
  'yaxixtitle',
  [20,10,59],
  'footer').
```

Example 2:

```
barchart1('This is the header text',
  'Barchart title',
  'xaxistitle',
  [ ['element1','legend_element1',Color1,Color2,Pattern],
           ['element2','legend_element2'],
           ['element3','legend_element3'] ],
  'yaxixtitle',
  [20,10,59],
  'footer').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`BarchartTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xbarelement1`s. (basic_props:list/2)

`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`YVector` is a list of `yelements`. (basic_props:list/2)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

barchart1/9: PREDICATE

`barchart1(Header,BTitle,XTitle,XVector,YTitle,YVector,YMax,YMin,Footer)`

You can set the minimum and maximum limits of the `y` axis. Data outside the limits will not be plotted. Each limit, as you can check by looking at the `axis_limit/1` definition, is a number. If the argument is a variable the limit will be calculated from the data (i.e., if `YMax` value is YValueMax the maximum y axis limit will calculated using the largest data value).

Example:

```
barchart1('This is the header text',
  'Barchart title',
  'xaxistitle',
  [ ['element1','e1','Blue','Yellow','pattern1'],
      ['element2','e2','Turquoise','Plum','pattern5'],
```

```
    ['element3','e3','Turquoise','Green','pattern5'] ],
'yaxixtitle',
[20,10,59],
70,
_,
'footer').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`BTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xbarelement1`s. (basic_props:list/2)

`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`YVector` is a list of `yelement`s. (basic_props:list/2)

genbar1:axis_limit(YMax) (genbar1:axis_limit/1)

genbar1:axis_limit(YMin) (genbar1:axis_limit/1)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

percentbarchart1/7: PREDICATE

`percentbarchart1(Header,BTitle,XTitle,XVector,YTitle,YVector,Footer)`

The y axis maximum coordinate value is 100. The `x` axis limits are automatically worked out.

Example:

```
percentbarchart1('This is a special barchart to represent percentages',
  'Barchart with legend',
  'My xaxistitle',
  [ [1,'bar1','Blue','Yellow','pattern1'],
       [8,'bar2','MediumTurquoise','Plum','pattern5'] ],
  'My yaxixtitle',
  [80,10],
  'This is the footer text').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`BTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xbarelement1`s. (basic_props:list/2)

`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`YVector` is a list of `yelement`s. (basic_props:list/2)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

yelement/1: REGTYPE

```
yelement(Y) :-
        number(Y).
```

Y is the bar lenght, so it must be a numeric value.

Both Prolog and Tcl-Tk support integers and floats. Integers are usually specified in decimal, but if the first character is 0 the number is read in octal (base 8), and if the first two characters are 0x, the number is read in hexadecimal (base16). Float numbers may be specified using most of the forms defined for ANSI C, including the following examples:

- 9.56
- 5.88e-2
- 5.1E2

Note: Be careful when using floats. While 8. or 7.e4 is interpreted by Tcl-tk as 8.0 and 7.0e4, Prolog will not read them as float numbers. Example:

```
?- number(8.e+5).
{SYNTAX ERROR: (lns 130-130) , or ) expected in arguments
number ( 8
** here **
. e + 5 ) .
}

no
?- number(8.).
{SYNTAX ERROR: (lns 138-138) , or ) expected in arguments
number ( 8
** here **
. ) .
}

no

?- number(8.0e+5).

yes
?- number(8.0).

yes
```

Precision: Tcl-tk internally represents integers with the C type `int`, which provides at least 32 bits of precision on most machines. Since Prolog integers can (in some implementations) exceed 32 bits but the precision in Tcl-tk depends on the machine, it is up to the progammer to ensure that the values fit into the maximum precision of the machine for integers. Real numbers are represented with the C type `double`, which is usually represented with 64-bit values (about 15 decimal digits of precision) using the IEEE Floating Point Standard.

Conversion: If the list is composed by integers and floats, Tcl-tk will convert integers to floats.

axis_limit/1: REGTYPE

```
axis_limit(X) :-
        number(X).
```

```
axis_limit(_1).
```

This type is defined in order to set the minimum and maximum limits of the axes. Data outside the limits will not be plotted. Each limit, is a number or a variable. If the argument is not a number the limit will be calculated from the data (i.e., if YMax value is `Var` the maximum y axis limit will be calculated using the largest data value).

header/1: REGTYPE

Usage: `header(X)`

– *Description:* `X` is a text (an atom) describing the header of the graph.

title/1: REGTYPE

Usage: `title(X)`

– *Description:* `X` is a text (an atom) to be used as label, usually not very long.

footer/1: REGTYPE

Usage: `footer(X)`

– *Description:* `X` is a text (an atom) describing the footer of the graph.

178.3 Documentation on internals (genbar1)

xbarelement1/1: REGTYPE

```
xbarelement1([XValue,LegendElement]) :-
        atomic(XValue),
        atomic(LegendElement).
xbarelement1([XValue,LegendElement,ForegColor,BackgColor,SPattern]) :-
        atomic(XValue),
        atomic(LegendElement),
        color(ForegColor),
        color(BackgColor),
        pattern(SPattern).
```

Defines the attributes of the bar.

`XValue` bar label. Although `XValue` values may be numbers, the will be treated as labels. Different elements with the same label will produce different bars.

`LegendElement`
Legend element name. It may be a number or an atom and equal or different to the XValue. Every `LegendElement` value of the list must be unique.

`ForegColor`
It sets the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`BackgColor`
It sets the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`SPattern` It sets the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.

178.4 Known bugs and planned improvements (`genbar1`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

179 Barchart widgets - 2

Author(s): Isabel Martín García.

This module defines predicates which show barchart widgets. The three predicates exported by this module plot two-variable data as regular bars in a window. They all share the following features:

- Numeric values for the x axis are needed, otherwise it does not work properly. See `xbarelement2/1` definition type.
- The bar position is proportional to the x-coordinate value.
- The user can either select the appearance of the bars (background color, foreground color and stipple style) or not. See the `xbarelement2/1` type definition. Thus, the user can call each predicate in two ways.
- The bar chart has a legend and one entry (symbol and label) per bar.
- If you do not want to display text in the elements header, barchart title, x axis title, y axis title or footer, simply type `''` as the value of the argument.
- The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception `error2` will be thrown. If the vectors contain elements but are not correct, the exception `error1` or `error3` will be thrown, depending on the error type. `error1` means that `XVector` and `YVector` does not contain the same number of elements and `error3` indicates that not all the `XVector` elements contain a correct number of attributes .

Particular features will be pointed out in the corresponding predicate.

179.1 Usage and interface (genbar2)

- **Library usage:**
  `:- use_module(library(genbar2)).`
- **Exports:**
  - *Predicates:*
    `barchart2/7`, `barchart2/11`, `percentbarchart2/7`.
  - *Regular Types:*
    `xbarelement2/1`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/color_pattern`, `chartlib/test_format`, `chartlib/install_utils`, `lists`, `random/random`.

179.2 Documentation on exports (genbar2)

barchart2/7: PREDICATE

`barchart2(Header,BarchartTitle,XTitle,XVector,YTitle,YVector,Footer)`

The maximum and minimum limits for axes are determined from the data.

Example:

```
barchart2('This is the header text',
  'Barchart with legend',
  'My xaxistitle',
  [ [1,'bar1','Blue','Yellow','pattern1'],
       [2,'bar2','MediumTurquoise','Plum','pattern5'] ],
  'My yaxixtitle',
  [20,10],
  'This is the footer text').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`BarchartTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xbarelement2`s. (basic_props:list/2)

`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`YVector` is a list of `yelements`. (basic_props:list/2)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

barchart2/11: PREDICATE

`barchart2(Header,BT,XT,XVector,XMax,XMin,YT,YVector,YMax,YMin,Footer)`

You can set the minimum and maximum limits of the axes. Data outside the limits will not be plotted. Each limit, as you can check looking at the `axis_limit/1` definition, is a number. If the argument is a variable the limit will be calculated from the data (i.e., if `YMax` value is YValueMax the maximum y axis limit will calculated using the largest data value).

Example:

```
barchart2('This is the header text',
  'Barchart with legend',
  'My xaxistitle',
  [ [1,'bar1',Color1,Color2,Pattern1],
        [2,'bar2',Color3,Color4,Pattern2] ],
  10,
  -10,
  'My yaxixtitle',
  [20,10],
  100,
  -10,
  'The limits for the axes are set by the user').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`BT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xbarelement2`s. (basic_props:list/2)

genbar1:axis_limit(XMax) (genbar1:axis_limit/1)

genbar1:axis_limit(XMin) (genbar1:axis_limit/1)

`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVector` is a list of `yelements`. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

percentbarchart2/7: PREDICATE

`percentbarchart2(Header,BTitle,XTitle,XVector,YTitle,YVector,Footer)`

The y axis maximum coordinate value is 100. The x axis limits are autoarrange.

Example:

```
percentbarchart2('This is a special barchart to represent percentages',
  'Barchart with legend',
  'My xaxistitle',
  [ [1,'bar1','Blue','Yellow','pattern1'],
          [2,'bar2','MediumTurquoise','Plum','pattern5'] ],
  'My yaxixtitle',
  [80,10],
  'This is the footer text').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`BTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVector` is a list of `xbarelement2`s. (basic_props:list/2)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVector` is a list of `yelements`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

xbarelement2/1: REGTYPE

```
xbarelement2([XValue,LegendElement]) :-
        number(XValue),
        atomic(LegendElement).
xbarelement2([XValue,LegendElement,ForegColor,BackgColor,SPattern]) :-
        number(XValue),
        atomic(LegendElement),
        color(ForegColor),
        color(BackgColor),
        pattern(SPattern).
```

Defines the attributes of the bar.

`XValue` x-coordinate position of the bar. Different elements with the same abscissas will produce overlapped bars.

`LegendElement`
Element legend name. It may be a number or an atom and equal or different to the XValue. Every `LegendElement` value of the list must be unique.

`ForegColor`
: Is the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`BackgColor`
: Is the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`SPattern`
: Is the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.

180 Depict barchart widgets - 3

Author(s): Isabel Martín García.

This module defines predicates which depict barchart widgets. The three predicates exported by this module plot two-variable data as regular bars in a window and are similar to those exported in the genbar1 module except in that these defined in this module do not display a legend. Thus, not all the argument types are equal.

The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception `error2` will be thrown. If the vectors contain elements but are not correct, the exception `error1` or `error3` will be thrown, depending on the error type. `error1` means that `XVector` and `YVector` do not contain the same number of elements and `error3` indicates that not all the `XVector` elements contain a correct number of attributes .

180.1 Usage and interface (genbar3)

- **Library usage:**
  `:- use_module(library(genbar3)).`
- **Exports:**
  - *Predicates:*
    `barchart3/7`, `barchart3/9`, `percentbarchart3/7`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/color_pattern`, `chartlib/test_format`, `chartlib/install_utils`, `lists`, `random/random`.

180.2 Documentation on exports (genbar3)

barchart3/7: PREDICATE

`barchart3(Header,BarchartTitle,XTitle,XVector,YTitle,YVector,Footer)`

As we mentioned in the above paragraph, this predicate is comparable to `barchart1/8` except in the `XVector` argument type.

Example:

```
barchart3('This is the header text',
  'Barchart without legend',
  'My xaxistitle',
  [['bar1'],['bar2']],
  'My yaxixtitle',
  [20,10],
  'This is the footer text').
```

Usage:

- *The following properties should hold at call time:*

  `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

  `BarchartTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

  `XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xbarelement3`s. (basic_props:list/2)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVector` is a list of `yelements`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

barchart3/9: PREDICATE

`barchart3(Header,BTitle,XTitle,XVector,YTitle,YVector,YMax,YMin,Footer)`

As we mentioned, this predicate is quite similar to the `barchart1/10` except in the `XVector` argument type, because the yielded bar chart lacks of legend.

Example:

```
barchart3('This is the header text',
  'Barchart without legend',
  'My xaxistitle',
  [['bar1'],['bar2']],
  'My yaxixtitle',
  30,
  5,
  [20,10],
  'This is the footer text').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`BTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVector` is a list of `xbarelement3`s. (basic_props:list/2)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVector` is a list of `yelements`. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

percentbarchart3/7: PREDICATE

`percentbarchart3(Header,BTitle,XTitle,XVector,YTitle,YVector,Footer)`

The y axis maximum coordinate value is 100. The x axis limits are autoarrange.

Example:

```
percentbarchart3('This is a special barchart to represent percentages',
          'Barchart without legend',
          'My xaxistitle',
          [ ['pr1','Blue','Yellow','pattern1'],
            ['pr2','MediumTurquoise','Plum','pattern5'] ],
          'My yaxixtitle',
          [80,10],
          'This is the footer text').
```

Usage:

– *The following properties should hold at call time:*
`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`BTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVector` is a list of `xbarelement3`s. (basic_props:list/2)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVector` is a list of `yelement`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

180.3 Documentation on internals (genbar3)

xbarelement3/1: REGTYPE

```
xbarelement3([XValue]) :-
        atomic(XValue).
xbarelement3([XValue,ForegColor,BackgColor,StipplePattern]) :-
        atomic(XValue),
        color(ForegColor),
        color(BackgColor),
        pattern(StipplePattern).
```

Defines the attributes of the bar.

`XValue` bar label. Although `XValue` values may be numbers, the will be treated as labels. Different elements with the same label will produce different bars.

`ForegColor`
It sets the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`BackgColor`
It sets the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`SPattern` It sets the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.

181 Depict barchart widgets - 4

Author(s): Isabel Martín García.

This module defines predicates which depict barchart widgets. The three predicates exported by this module plot two-variable data as regular bars in a window and are similar to those exported in genbar2 module except in that those defined in this module doesn't display a legend. Thus, the user does not have to define legend element names.

The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception `error2` will be thrown. If the vectors contains elements but are not correct, the exception `error1` or `error3` will be thrown, depending on the error type. `error1` means that `XVector` and `YVector` do not contain the same number of elements and `error3` indicates that not all the `XVector` elements contain a correct number of attributes .

181.1 Usage and interface (genbar4)

- **Library usage:**
  `:- use_module(library(genbar4)).`
- **Exports:**
  - *Predicates:*
    `barchart4/7`, `barchart4/11`, `percentbarchart4/7`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/color_pattern`, `chartlib/test_format`, `chartlib/install_utils`, `lists`, `random/random`.

181.2 Documentation on exports (genbar4)

barchart4/7: PREDICATE

`barchart4(Header,BarchartTitle,XTitle,XVector,YTitle,YVector,Footer)`

As we mentioned in the above paragraph, this predicate is comparable to `barchart2/8` except in the `XVector` argument type.

Example:

```
barchart4('This is the header text',
  'Barchart without legend',
  'My xaxistitle',
  [[2],[5],[6]],
  'My yaxixtitle',
  [20,10,59],
  'Numeric values in the xaxis').
```

Usage:

- *The following properties should hold at call time:*

  `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

  `BarchartTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

  `XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xbarelement4`s. (basic_props:list/2)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVector` is a list of `yelements`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

barchart4/11: PREDICATE

`barchart4(Hder,BT,XT,XVector,XMax,XMin,YT,YVector,YMax,YMin,Fter)`

As we stated before, this predicate is quite similar to `barchart2/10` except in the following aspects:

- The `XVector` argument type, because the yielded bar chart lacks the legend.
- The user can set limits for both `x` axis and `y` axis.

Example:

```
barchart4('This is the header text, you can write a graph description',
  'Barchart without legend',
  'My xaxistitle',
  [[2,'Blue','Yellow','pattern1'],
       [20,'MediumTurquoise','Plum','pattern5'],
       [30,'MediumTurquoise','Green','pattern5']],
  50,
  -10,
  'My yaxixtitle',
  [20,10,59],
  100,
  -10,
  'Numeric values in the xaxis').
```

Usage:

– *The following properties should hold at call time:*

`Hder` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`BT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVector` is a list of `xbarelement4`s. (basic_props:list/2)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVector` is a list of `yelements`. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`Fter` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

percentbarchart4/7: PREDICATE

`percentbarchart4(Header,BTitle,XTitle,XVector,YTitle,YVector,Footer)`

The y axis maximum coordinate value is 100. The x axis limits are automatically worked out. This predicate is useful when the bar height represents percentages.

Example:

```
percentbarchart4('This is the header text',
  'Barchart without legend',
  'My xaxistitle',
  [[2,'Blue','Yellow','pattern1'],[5,'Yellow','Plum','pattern5'],
       [6,'MediumTurquoise','Green','pattern5']],
  'My yaxixtitle',
  [20,10,59],
  'Numeric values in the xaxis').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`BTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xbarelement4`s. (basic_props:list/2)

`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`YVector` is a list of `yelement`s. (basic_props:list/2)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

181.3 Documentation on internals (genbar4)

xbarelement4/1: REGTYPE

Defines the attributes of the bar.

`XValue` x-coordinate position of the bar. Different elements with the same abscissas will produce overlapped bars.

`ForegColor`
It sets the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`BackgColor`
It sets the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`SPattern` It sets the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.

182 Depic line graph

Author(s): Isabel Martín García.

This module defines predicates which depict line graph and scatter graph widgets. All eigth predicates exported by this module plot two-variable data. Each point is defined by its X-Y coordinate values. A dataset is defined by two lists xvector and yvector, which contain the points coordinates. As you might guess, the values placed in the the same position in both lists are the coordinates of a point. They both share the following features:

- Numeric values for vector elements are needed. We'll use two vectors to represent the X-Y coordinates of each set of plotted data, but in this case every dataset shares the X-vector, i.e., x-coordinate of points with the same index[1] in different datasets is the same. Thus, the numbers of points in each yvector must be equal to the number of points in the xvector.
- The active element color is navyblue, which means that when you select a legend element, the corresponding line or point dataset turns into navyblue.
- The user can either select the appearance of the lines and/or points of each dataset or not. If not, the system will choose the colors for the lines and the points among the available ones in accordance with the plot background color and it will also set the points size and symbol to the default. If the plot background color is black, the system will choose a lighter color, and the system will select a darker color when the plot background color is white. Thus, the user can define the appearanse attributes of each dataset in four different ways. Take a look at the `attributes/1` type definition and see the examples to understand it clearly.
- The graph has a legend and one entry (symbol and label) per dataset.
- If you do not want to display text in the element header, barchart title, xaxis title, yaxis title or footer, simply give `''` as the value of the argument.
- The predicates check whether the format of the arguments is correct as well. The testing process involves some verifications. If one or both vectors are empty, the exception `error2` will be thrown. If the vectors contains elements but are not correct, the exception `error4` will be thrown.

The names of the line graph predicates begin with **graph_** and those corresponding to the scatter graph group begin with **scattergraph_**.

[1] It should be pointed out that I am refering to an index as the position of an element in a list.

182.1 Usage and interface (gengraph1)

- **Library usage:**
  `:- use_module(library(gengraph1)).`
- **Exports:**
  - *Predicates:*
    `graph_b1/9`, `graph_b1/13`, `graph_w1/9`, `graph_w1/13`, `scattergraph_b1/8`, `scattergraph_b1/12`, `scattergraph_w1/8`, `scattergraph_w1/12`.
  - *Regular Types:*
    `vector/1`, `smooth/1`, `attributes/1`, `symbol/1`, `size/1`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/bltclass`, `chartlib/genbar1`, `chartlib/color_pattern`, `chartlib/test_format`, `chartlib/install_utils`, `lists`, `random/random`.

182.2 Documentation on exports (gengraph1)

graph_b1/9: PREDICATE

`graph_b1(Header,GTitle,XTitle,XVector,YTitle,YVectors,LAtts,Footer,Smooth)`

Besides the features mentioned at the begining of the chapter, the displayed graph generated when calling this predicate has the following distinguishing characteristics:

- The plotting area background color is black.
- The cross hairs color is white.
- The axes limits are determined from the data.

Example:

```
graph_b1('This is the header text',
  'Graph_title',
  'xaxistitle',
  [20,10,59],
  'yaxixtitle',
  [ [10,35,40],[25,50,60] ],
  [ ['element1','Blue','Yellow','plus',6],['element2',Outline,Color] ],
  'footer',
  'linear').
```

Usage:

- *The following properties should hold at call time:*

  `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

  `GTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

  `XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

  gengraph1:vector(XVector) (gengraph1:vector/1)

  `YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

  `YVectors` is a list of `vectors`. (basic_props:list/2)

  `LAtts` is a list of `attributess`. (basic_props:list/2)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
gengraph1:smooth(Smooth) (gengraph1:smooth/1)

graph_b1/13: PREDICATE

`graph_b1(Header,GT,XT,XV,XMax,XMin,YT,YVs,YMax,YMin,LAtts,Footer,Smooth)`

The particular features related to this predicate are described below:

- The plotting area background color is black.
- The cross hairs color is white.
- You can set the minimum and maximum limits of the axes. Data outside the limits will not be plotted.

Example:

```
graph_b1('This is the header text',
  'Graph_title',
  'xaxistitle',
  [20,10,59],
  50,
  _,
  'yaxixtitle',
  [[10,35,40],[25,50,60]],
  50,
  _,
  [['line1','circle',4],['line2',OutlineColor,Color]],
  'footer',
  'step').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
gengraph1:vector(XV) (gengraph1:vector/1)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vectors`. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`LAtts` is a list of `attributess`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
gengraph1:smooth(Smooth) (gengraph1:smooth/1)

graph_w1/9: PREDICATE

`graph_w1(Header,GTitle,XTitle,XVector,YTitle,YVectors,LAtts,Footer,Smooth)`

This predicate is quite similar to `graph_b1/9`. The differences lies in the plot background color and in the cross hairs color, which are white and black respectively.

Example:

```
graph_w1('This is the header text',
  'Graph_title',
  'xaxistitle',
  [20,10,40,50],
  'yaxixtitle',
  [ [10,35,40,50],[25,20,60,40] ],
  [['line1','Blue','DarkOrchid'],['line2','circle',3]],
  'footer',
  'quadratic').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
gengraph1:vector(XVector) (gengraph1:vector/1)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVectors` is a list of `vector`s. (basic_props:list/2)
`LAtts` is a list of `attributes`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
gengraph1:smooth(Smooth) (gengraph1:smooth/1)

graph_w1/13: PREDICATE

`graph_w1(Header,GT,XT,XV,XMax,XMin,YT,YVs,YMax,YMin,LAtts,Footer,Smooth)`

This predicate is quite similar to `graph_b1/13`, the differences between them are listed below:

- The plotting area background color is white.
- The cross hairs color is black.

Example:

```
graph_w1('This is the header text',
  'Graph_title',
  'xaxistitle',
  [20,10,59],
  100,
  10,
  'yaxixtitle',
  [[10,35,40],[25,20,60]],
  _,
  _,
  [['element1','Blue','Yellow'],['element2','Turquoise','Plum']],
  'footer',
  'quadratic').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

gengraph1:vector(XV) (gengraph1:vector/1)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vector`s. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`LAtts` is a list of `attributes`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
gengraph1:smooth(Smooth) (gengraph1:smooth/1)

scattergraph_b1/8: PREDICATE

`scattergraph_b1(Header,GTitle,XTitle,XVector,YTitle,YVectors,PAtts,Footer)`

Apart from the features brought up at the beginning of the chapter, the scatter graph displayed invoking this predicate has the following characteristics:

- The plotting area background color is black.
- The cross hairs color is white.
- The axes limits are determined from the data.

Example:

```
scattergraph_b1('This is the header text',
  'Graph_title',
  'xaxistitle',
  [10,15,20],
  'yaxixtitle',
  [[10,35,20],[15,11,21]],
  [['element1','Blue','Yellow'],['element2','Turquoise','Plum']],
  'footer').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
gengraph1:vector(XVector) (gengraph1:vector/1)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVectors` is a list of `vector`s. (basic_props:list/2)
`PAtts` is a list of `attributes`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_b1/12: PREDICATE

`scattergraph_b1(Header,GT,XT,XV,XMax,XMin,YT,YVs,YMax,YMin,PAtts,Footer)`

The particular features related to this predicate are described below:

- The plotting area background color is black.
- The cross hairs color is white.

- You can set the minimum and maximum limits of the axes. Data outside the limits will not be plotted.

Example:

```
scattergraph_b1('This is the header text',
  'Graph_title',
  'xaxistitle',
  [20,10,59],
  50,
  _,
  'yaxixtitle',
  [[10,35,40],[25,50,60]],
  50,
  _,
  [['point dataset1','Blue','Yellow'],['point dataset2']],
  'footer').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
gengraph1:vector(XV) (gengraph1:vector/1)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vectors`. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`PAtts` is a list of `attributess`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_w1/8: PREDICATE

`scattergraph_w1(Header,GT,XT,XVector,YT,YVectors,PAtts,Footer)`

This predicate is quite similar to `scattergraph_b1/8` except in the following:

- The plotting area background color is black.
- The cross hairs color is white.
- If the user does not fix the points colors, they will be chosen among the lighter ones.

Example:

```
scattergraph_w1('This is the header text',
  'Graph_title',
  'xaxistitle',
  [20,10,59],
  'yaxixtitle',
  [[10,35,40],[25,20,60]],
  [['e1','Blue','Green'],['e2','MediumVioletRed','Plum']],
  'footer').
```

Usage:

– *The following properties should hold at call time:*
`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
gengraph1:vector(XVector) (gengraph1:vector/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVectors` is a list of `vectors`. (basic_props:list/2)
`PAtts` is a list of `attributess`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_w1/12: PREDICATE

`scattergraph_w1(Header,GT,XT,XV,XMax,XMin,YT,YVs,YMax,YMin,PAtts,Footer)`

This predicate is quite similar to `scattergraph1_b1/13`, the differences between them are listed below:

- The plotting area background color is white.
- The cross hairs color is black.

Example:

```
scattergraph_w1('This is the header text',
  'Graph_title',
  'xaxistitle',
  [20,10,59],
  150,
  5,
  'yaxixtitle',
  [[10,35,40],[25,20,60]],
  -,
  -10,
  [['e1','Blue','Yellow'],['e2','MediumTurquoise','Plum']],
  'footer').
```

Usage:

– *The following properties should hold at call time:*
`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
gengraph1:vector(XV) (gengraph1:vector/1)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vectors`. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`PAtts` is a list of `attributess`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

vector/1: REGTYPE

```
vector(X) :-
        list(X,number).
```

The type vector defines a list of numbers (integers or floats).

smooth/1: REGTYPE

```
smooth(Smooth)
```

```
smooth(linear).
smooth(cubic).
smooth(quadratic).
smooth(step).
```

Specifies how connecting segments are drawn between data points. If `Smooth` is `linear`, a single line segment is drawn, connecting both data points. When `Smooth` is `step`, two line segments will be drawn, the first line is a horizontal line segment that steps the next X-coordinate and the second one is a vertical line, moving to the next Y-coordinate. Both `cubic` and `quadratic` generate multiple segments between data points. If `cubic`is used, the segments are generated using a cubic spline. If `quadratic`, a quadratic spline is used. The default is linear.

attributes/1: REGTYPE

```
attributes([ElementName]) :-
        atomic(ElementName).
attributes([ElementName,OutLine,Color]) :-
        atomic(ElementName),
        color(OutLine),
        color(Color).
attributes([ElementName,Symbol,Size]) :-
        atomic(ElementName),
        symbol(Symbol),
        size(Size).
attributes([ElementName,OutLine,Color,Symbol,Size]) :-
        atomic(ElementName),
        color(OutLine),
        color(Color),
        symbol(Symbol),
        size(Size).
```

Each line or point dataset in the graph has its own attributes, which are defined by this type. The name of the dataset, specified in the `ElementName` argument, may be either a number or an atom. The second argument is the color of a thin line around each point in the dataset and the Color argument is the points and lines color. Both `OutLine` and `Color` must be a valid color (see available values in `color/1`), otherwise a random color according to the plot background color will be selected. The `Symbol` must be a valid symbol and the `Size` must be a number. Be careful if you want to especify the `Symbol` and the `Size`, otherwise the predicate will not work as you expect. If you don't select a symbol and a size for a dataset the default values will be square and 1 pixel.

symbol/1: REGTYPE

```
symbol(Symbol)
```

```
symbol(square).
symbol(circle).
symbol(diamond).
symbol(plus).
symbol(cross).
symbol(splus).
symbol(scross).
symbol(triangle).
```

`Symbol` stands for the shape of the points whether in scatter graphs or in line graphs.

size/1: REGTYPE

```
size(Size)
```

```
size(Size) :-
        number(Size).
```

`Size` stands for the size in pixels of the points whether in scatter graphs or in line graphs.

183 Line graph widgets

Author(s): Isabel Martín García.

This module defines predicates which show line graph widgets. All eight predicates exported by this module plot two-variable data. Each point is defined by its X-Y coordinate values. Every predicate share the following features:

- A dataset is defined by three lists xvector, yvector and attributes. The arguments named `XVectors` (or `XVs`), `YVectors` (or `YVs`) and `LAtts`[1] contain this information. Those arguments must be lists whose elements are also lists. The first dataset is defined by the firts element of the three lists, the second dataset is defined by the second element of the three lists and so on.
- Numeric values for the vector elements are needed. We will use two vectors to represent the X-Y coordinates of each set of data plotted. In these predicates the vectors can have different number of points. However, the number of elements in xvector and yvector that pertain to a certain dataset must be, obviously, equal.
- The active line color is blue, which means that when you select a legend element, the corresponding line turns into navyblue.
- The user can either select the appearance for the lines and the points or not. See the `attributes/1` type definition. Thus, the user can call each predicate in different ways ways.
- The graph has a legend and one entry (symbol and label) per dataset.
- If you do not want to display text in the elements header, barchart title, xaxis title, yaxis title or footer, simply give `''` as the value of the argument.
- The predicates check whether the format of the arguments is correct as well. The testing process involves some verifications. If one or both vectors are empty, the exception `error2` will be thrown. If the vectors contains elements but are not correct, the exception `error4` will be thrown.

183.1 Usage and interface (gengraph2)

- **Library usage:**
  `:- use_module(library(gengraph2)).`
- **Exports:**
  - *Predicates:*
    `graph_b2/9`, `graph_b2/13`, `graph_w2/9`, `graph_w2/13`, `scattergraph_b2/8`, `scattergraph_b2/12`, `scattergraph_w2/8`, `scattergraph_w2/12`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/gengraph1`, `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/color_pattern`, `chartlib/test_format`, `chartlib/install_utils`, `lists`, `random/random`.

183.2 Documentation on exports (gengraph2)

[1] In scatter graphs the attibute that contains the features of a point dataset is `PAtts`.

graph_b2/9: PREDICATE

`graph_b2(Header,GTitle,XTitle,XVectors,YTitle,YVectors,LAtts,Footer,Sm)`

Besides the features mentioned at the begining of the module chapter, the displayed graph generated calling this predicate has the following distinguish characteristics:

- The plotting area background color is black.
- The cross hairs color is white.
- The axis limits are determined from the data.

Example:

```
graph_b2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[20,30,59],[25,50]],
  'yaxixtitle',
  [[10,35,40],[25,50]],
  [['line1','Blue','Yellow'],['line2']],
  'footer',
  'natural').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`GTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVectors` is a list of `vectors`. (basic_props:list/2)

`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`YVectors` is a list of `vectors`. (basic_props:list/2)

`LAtts` is a list of `attributess`. (basic_props:list/2)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

gengraph1:smooth(Sm) (gengraph1:smooth/1)

graph_b2/13: PREDICATE

`graph_b2(Header,GT,XT,XVs,XMax,XMin,YT,YVs,YMax,YMin,LAtts,Footer,Smooth)`

In addition to the features brought up at the begining of the module chapter, this graph has the following:

- The plotting area background color is black.
- The cross hairs color is white.
- You can set the maximum and minimum values for the graph axes.

Example:

```
graph_b2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[20,10,59],[15,30,35]],
  50,
  _,
  'yaxixtitle',
  [[10,35,40],[25,50,60]],
```

```
50.5,
_,
[['line1','Blue','Yellow'],['line','MediumTurquoise','Plum']],
'footer',
'step').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVs` is a list of `vector`s. (basic_props:list/2)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vector`s. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`LAtts` is a list of `attributes`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
gengraph1:smooth(Smooth) (gengraph1:smooth/1)

graph_w2/9: PREDICATE

`graph_w2(Header,GT,XT,XVectors,YTitle,YVectors,LAtts,Footer,Smooth)`

This predicate is quite similar to `graph_b2/9`. The difference lies in the graph appearance, as you can see below.

- The plotting area background color is white.
- The cross hairs color is black.

Example:

```
graph_w2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[10,30,59],[25,50]],
  'yaxixtitle',
  [[10,35,40],[25,40]],
  [['element1','Blue','DarkOrchid'],['element2','DarkOliveGreen',
    'Firebrick']],
  'footer',
  'natural').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVectors` is a list of `vector`s. (basic_props:list/2)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`YVectors` is a list of `vectors`. (basic_props:list/2)
`LAtts` is a list of `attributess`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
gengraph1:smooth(Smooth) (gengraph1:smooth/1)

graph_w2/13: PREDICATE

`graph_w2(Header,GT,XT,XV,XMax,XMin,YT,YVs,YMax,YMin,LAtts,Footer,Smooth)`

This predicate is comparable to `graph_b2/13`. The differences lie in the plot background color and in the cross hairs color, wich are white and black respectively.

Example:

```
graph_w2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[10,30,59],[10,35,40]],
  80,
  _,
  'yaxixtitle',
  [[10,35,40],[25,50,60]],
  50,
  _,
  [['element1','Blue','Green'],['element2','Turquoise','Black']],
  'footer',
  'linear').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XV` is a list of `vectors`. (basic_props:list/2)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vectors`. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`LAtts` is a list of `attributess`. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)
gengraph1:smooth(Smooth) (gengraph1:smooth/1)

scattergraph_b2/8: PREDICATE

`scattergraph_b2(Header,GT,XT,XVectors,YT,YVectors,PAtts,Footer)`

Apart from the features brought up at the beginning of the chapter, the scatter graph displayed when invoking this predicate has the following features:

- The plotting area background color is black.

- The cross hairs color is white.
- The axis limits are determined from the data.

Example:

```
scattergraph_b2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[10,15,20],[8,30,40]],
  'yaxixtitle',
  [[10,35,20],[15,11,21]],
  [['element1','Blue','Yellow'],['element2','MediumTurquoise','Plum']],
  'footer').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVectors` is a list of `vector`s. (basic_props:list/2)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVectors` is a list of `vector`s. (basic_props:list/2)
`PAtts` is a list of `attributes`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_b2/12: PREDICATE

`scattergraph_b2(Header,GT,XT,XVs,XMax,XMin,YT,YVs,YMax,YMin,PAtts,Footer)`

The particular features related to this predicate are described below:

- The plotting area background color is black.
- The cross hairs color is white.
- You can set the minimum and maximum limits of the axes. Data outside the limits will not be plotted.

Example:

```
scattergraph_b2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[20,30,50],[18,40,59]],
  50,
  _,
  'yaxixtitle',
  [[10,35,40],[25,50,60]],
  50,
  _,
  [['point dataset1'],['point dataset2']],
  'footer').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVs` is a list of `vector`s. (basic_props:list/2)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vector`s. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`PAtts` is a list of `attributes`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_w2/8: PREDICATE

`scattergraph_w2(Header,GTitle,XTitle,XVs,YTitle,YVs,PAtts,Footer)`

This predicate is quite similar to `scattergraph_w1/8` except in the following:

- The plotting area background color is black.
- The cross hairs color is white.
- If the user do not provide the colors of the points, they will be chosen among the lighter ones.

Example:

```
scattergraph_w2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[20,30,40,15,30,35,20,30]],
  'yaxixtitle',
  [[10,30,40,25,20,25,20,25]],
  [['set1','cross',4]],
  'footer').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
gengraph1:vector(XVs) (gengraph1:vector/1)
`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vector`s. (basic_props:list/2)
`PAtts` is a list of `attributes`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

scattergraph_w2/12: PREDICATE

`scattergraph_w2(Header,GT,XT,XVs,XMax,XMin,YT,YVs,YMax,YMin,PAtts,Footer)`

This predicate is comparable to `scattergraph_w2/13`, the differences between them are listed below:

- The plotting area background color is white.
- The cross hairs color is black.

Example:

```
scattergraph_w2('This is the header text',
  'Graph_title',
  'xaxistitle',
  [[20,10,59],[15,30,50]],
  150,
  5,
  'yaxixtitle',
  [[10,35,40],[25,20,60]],
  _,
  -10,
  [['e1','Blue','Yellow'],['e2','MediumTurquoise','Plum']],
  'footer').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
`GT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`XVs` is a list of `vector`s. (basic_props:list/2)
genbar1:axis_limit(XMax) (genbar1:axis_limit/1)
genbar1:axis_limit(XMin) (genbar1:axis_limit/1)
`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
`YVs` is a list of `vector`s. (basic_props:list/2)
genbar1:axis_limit(YMax) (genbar1:axis_limit/1)
genbar1:axis_limit(YMin) (genbar1:axis_limit/1)
`PAtts` is a list of `attributes`s. (basic_props:list/2)
`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

184 Multi barchart widgets

Author(s): Isabel Martín García.

This module defines predicates which show barchart widgets. These bar charts are somewhat different from the bar charts generated by the predicates in modules genbar1, genbar2, genbar3 and genbar4. Predicates in the present module show different features of each dataset element in one chart at the same time. Each bar chart element is a group of bars, and the element features involve three vectors defined as follows:

- xvector is a list containing the names (atoms) of the bars (n elements). Each bar group will be displayed at uniform intervals.
- yvector is a list that contains m sublists, each one is composed of n elements. The i-sublist contains the y-values of the i-BarAttribute element for all of the XVector elements.
- bar_attributtes is a list containing the appearance features of the bars (m elements). Each element of the list can be partial or complete, which means that you can define as bar attributes only the element name or by setting the element name, its background and foreground color and its stipple pattern.

Other relevant aspects about this widgets are:

- If you don't want to display text in the elements header, barchart title, xaxis title, yaxis title or footer, simply type `''` as the value of the argument.
- The bar chart has a legend, and one entry (symbol and label) per feature group bar.
- The user can either select the appearance of the bars (background color, foreground color and stipple style) or not. See the multibar_attribute type definition.
- Data points can have their bar segments displayed in one of the following modes: stacked, aligned, overlapped or overlayed. They user can change the mode clicking in the checkboxes associated to each mode.
- The predicates test whether the format of the arguments is correct. If one or both vectors are empty, the exception `error2` will be thrown. If the vectors contains elements but are not correct, the exception `error5` or `error6` will be thrown, depending on what is incorrect. `error5` means that `XVector` and each element of `YVector` do not contain the same number of elements or that `YVector` and `BarsAtt` do not contain the same number of elements, while `error6` indicates that not all the `BarsAtt` elements contain a correct number of attributes.

The examples will help you to understand how these predicates should be called.

184.1 Usage and interface (`genmultibar`)

- **Library usage:**
  `:- use_module(library(genmultibar)).`
- **Exports:**
  - *Predicates:*
    `multibarchart/8`, `multibarchart/10`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/color_pattern`, `chartlib/test_format`, `chartlib/install_utils`, `lists`, `random/random`.

184.2 Documentation on exports (genmultibar)

multibarchart/8: PREDICATE

`multibarchart(Header,BTitle,XTitle,XVector,YTitle,BarsAtts,YVector,Footer)`

The x axis limits are autoarrange. The user can call the predicate in two ways. In the first example the user sets the appearance of the bars, in the second one the appearance features will be chosen by the library.

Example1:

```
multibarchart('This is the Header text',
'My BarchartTitle',
'Processors',
['processor1','processor2','processor3','processor4'],
'Time (seconds)',
[['setup time','MediumTurquoise','Plum','pattern2'],
    ['sleep time','Blue','Green','pattern5'],
    ['running time','Yellow','Plum','pattern1']],
[[20,30,40,50],[10,8,5,35],[60,100,20,50]],
'This is the Footer text').
```

Example2:

```
multibarchart('This is the Header text',
'My BarchartTitle',
'Processors',
['processor1','processor2','processor3','processor4'],
'Time (seconds)',
[['setup time'],['sleep time'],['running time']],
[[20,30,40,50],[10,8,5,35],[60,100,20,50]],
'This is the Footer text').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`BTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xelements`. (basic_props:list/2)

`YTitle` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`BarsAtts` is a list of `multibar_attributes`. (basic_props:list/2)

`YVector` is a list of `yelements`. (basic_props:list/2)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

multibarchart/10: PREDICATE

`multibarchart(Header,BT,XT,XVector,YT,BAtts,YVector,YMax,YMin,Footer)`

This predicate is quite similar to `multibarchart/8`, except in that you can choose limits in the y axis. The part of the bars placed outside the limits will not be plotted.

Example2:

```
multibarchart('This is the Header text',
'My BarchartTitle',
'Processors',
```

```
['processor1','processor2','processor3','processor4'],
'Time (seconds)',
[['setup time'],['sleep time'],['running time']],
[[20,30,40,50],[10,8,5,35],[60,100,20,50]],
[80],
[0],
'This is the Footer text').
```

Usage:

– *The following properties should hold at call time:*

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

`BT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`XVector` is a list of `xelements`. (basic_props:list/2)

`YT` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`BAtts` is a list of `multibar_attributes`. (basic_props:list/2)

`YVector` is a list of `yelements`. (basic_props:list/2)

genbar1:axis_limit(YMax) (genbar1:axis_limit/1)

genbar1:axis_limit(YMin) (genbar1:axis_limit/1)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

184.3 Documentation on internals (`genmultibar`)

multibar_attribute/1: REGTYPE

```
multibar_attribute([LegendElement]) :-
        atomic(LegendElement).
multibar_attribute([LegendElement,ForegroundColor,BackgroundColor,StipplePatter
        atom(LegendElement),
        color(ForegroundColor),
        color(BackgroundColor),
        pattern(StipplePattern).
```

Defines the attributes of each feature bar along the different datasets.

LegendElement
: Legend element name. It may be a number or an atom. Every `LegendElement` value of the list must be unique.

`ForegroundColor`
: It sets the Foreground color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`BackgroundColor`
: It sets the Background color of the bar. Its value must be a valid color, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a color chosen by the library.

`StipplePattern`
: It sets the stipple of the bar. Its value must be a valid pattern, otherwise the system will throw an exception. If the argument value is a variable, it gets instantiated to a pattern chosen by the library.

xelement/1: REGTYPE

```
xelement(Label) :-
        atomic(Label).
```

This type defines a dataset label. Although `Label` values may be numbers, the will be treated as atoms, So it will be displayed at uniform intervals along the x axis.

185 table_widget1 (library)

Author(s): Isabel Martín García.

In addition to the features explained in the introduction, the predicates exported by this module depict tables in which the font weight for the table elements is bold.

If the arguments are not in a correct format the exception `error8` will be thrown.

185.1 Usage and interface (`table_widget1`)

- **Library usage:**
  `:- use_module(library(table_widget1)).`
- **Exports:**
  - *Predicates:*
    `tablewidget1/4`, `tablewidget1/5`.
  - *Regular Types:*
    `table/1`, `image/1`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/test_format`, `chartlib/install_utils`, `lists`.

185.2 Documentation on exports (`table_widget1`)

tablewidget1/4: PREDICATE

`tablewidget1(Title,Header,ElementTable,Footer)`

Shows a regular table in a window. The user does not choose a background image.

Example:

```
tablewidget1('This is the title',
     'Header text',
     [['Number of processors','8'],['Average processors','95'],
         ['Average Tasks per fork','7.5']],
     'Footer text').
```

Usage:

- *The following properties should hold at call time:*
  `Title` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
  table_widget1:table(ElementTable) (table_widget1:table/1)
  `Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

tablewidget1/5: PREDICATE

`tablewidget1(Title,Header,ElementTable,Footer,BackgroundImage)`

Shows a regular table in a window. The user must set a background image. See the `image/1` type definition.

Example:

```
tablewidget1('This is the title',
     'Header text',
     [['Number of processors','8'],['Average processors','95'],
         ['Average Tasks per fork','7.5']],
     'Footer text',
     './images/rain.gif')
```

Usage:

– *The following properties should hold at call time:*

`Title` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

table_widget1:table(ElementTable) (table_widget1:table/1)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

table_widget1:image(BackgroundImage) (table_widget1:image/1)

table/1: REGTYPE

A table is a list of rows, each row must contain the same number of elements, otherwise the table wouldn't be regular and an exception will be thrown by the library. The rows list may not be empty.

```
table([X]) :-
        row(X).
table([X|Xs]) :-
        row(X),
        table(Xs).
```

image/1: REGTYPE

Some predicates allow the user to set the widget background image, whose is what this type is intended for. The user has to take into account the following restrictions:

- The image must be in gif format.
- The file path must be absolute.

185.3 Documentation on internals (`table_widget1`)

row/1: REGTYPE

```
row([X]) :-
        cell_value(X).
row([X|Xs]) :-
        cell_value(X),
        row(Xs).
```

Each row is a list of elements whose type is `cell_value/1`. A row cannot be an empty list, as you can see in the definition type.

row/1: REGTYPE

```
row([X]) :-
        cell_value(X).
row([X|Xs]) :-
        cell_value(X),
        row(Xs).
```

Each row is a list of elements whose type is `cell_value/1`. A row cannot be an empty list, as you can see in the definition type.

cell_value/1: REGTYPE

This type defines the possible values that a table element have. If any cell value is `''`, the cell will be displayed empty.

```
cell_value(X) :-
        atomic(X).
```

186 table_widget2 (library)

Author(s): Isabel Martín García.

In addition to the features explained in the introduction, predicates exported by this module display tables in which the font weight for the elements placed in the first row is bold. The remaining elements are in medium weight font.

If the arguments are not in a correct format the exception `error8` will be thrown.

186.1 Usage and interface (table_widget2)

- **Library usage:**
  `:- use_module(library(table_widget2)).`
- **Exports:**
  - *Predicates:*
    `tablewidget2/4`, `tablewidget2/5`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/table_widget1`, `chartlib/test_format`, `chartlib/install_utils`, `lists`.

186.2 Documentation on exports (table_widget2)

tablewidget2/4: PREDICATE

`tablewidget2(Title,Header,ElementTable,Footer)`

Shows a regular table in a window. The system sets a default background image for the widget.

Example:

```
tablewidget2('COM Features',
     'Extracted from "Inside COM" book ',
     [['Feature','Rich people','Bean Plants','C++','COM'],
       ['Edible','Yes','Yes','No','No'],
       ['Supports inheritance','Yes','Yes','Yes','Yes and No'],
       ['Can run for President','Yes','No','No','No']],
     'What do you think about COM?').
```

Usage:

- *The following properties should hold at call time:*
  `Title` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
  table_widget1:table(ElementTable) (table_widget1:table/1)
  `Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

tablewidget2/5: PREDICATE

`tablewidget2(Title,Header,ElementTable,Footer,BackgroundImage)`

This predicate and `tablewidget2/4` are quite similar, except that in the already one defined you must set the background image.

Example:

```
tablewidget2('COM Features',
     'Extracted from "Inside COM" book ',
     [['Feature','Rich people','Bean Plants','C++','COM'],
       ['Edible','Yes','Yes','No','No'],
       ['Supports inheritance','Yes','Yes','Yes','Yes and No'],
       ['Can run for President','Yes','No','No','No']],
     'What do you think about COM?',
     './images/rain.gif').
```

Usage:

– *The following properties should hold at call time:*

`Title` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

table_widget1:table(ElementTable) (table_widget1:table/1)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

table_widget1:image(BackgroundImage) (table_widget1:image/1)

187 table_widget3 (library)

Author(s): Isabel Martín García.

The predicates exported by this module display data in a regular table, as we brought up in the introduction. Both predicates have in common that the font weight for the elements placed in the first column is bold and the remaining elements are in medium font weight.

If the arguments are not in a correct format the exception `error8` will be thrown.

187.1 Usage and interface (`table_widget3`)

- **Library usage:**
  `:- use_module(library(table_widget3)).`
- **Exports:**
  - *Predicates:*
    `tablewidget3/4`, `tablewidget3/5`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/table_widget1`, `chartlib/test_format`, `chartlib/install_utils`, `lists`.

187.2 Documentation on exports (`table_widget3`)

tablewidget3/4: PREDICATE

`tablewidget3(Title,Header,ElementTable,Footer)`

Shows a regular table in a window. The user does not choose a background image.

Example:

```
tablewidget3('This is the title',
     'Header text',
     [['Number of processors','8'],['Average processors','95'],
       ['Tasks per fork','7.5']],
     'Footer text').
```

Usage:

- *The following properties should hold at call time:*
  `Title` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)
  `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)
  table_widget1:table(ElementTable) (table_widget1:table/1)
  `Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

tablewidget3/5: PREDICATE

`tablewidget3(Title,Header,ElementTable,Footer,BackgroundImage)`

Shows a regular table in a window. The user must set a background image.

Example:

```
tablewidget3('This is the title',
     'Header text',
     [['Number of processors','8'],['Average processors','95'],
         ['Average Tasks per fork','7.5']],
     'Footer text',
     './images/rain.gif')
```

Usage:

– *The following properties should hold at call time:*

`Title` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

table_widget1:table(ElementTable) (table_widget1:table/1)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

table_widget1:image(BackgroundImage) (table_widget1:image/1)

188 table_widget4 (library)

Author(s): Isabel Martín García.

In addition to the features explained in the introduction, predicates exported by this module display tables in which the font weight for the elements placed in the first row and column is bold. The remaining elements are in medium weight font.

If the arguments are not in a correct format the exception `error8` will be thrown.

188.1 Usage and interface (`table_widget4`)

- **Library usage:**
  `:- use_module(library(table_widget4)).`
- **Exports:**
  - *Predicates:*
    `tablewidget4/4`, `tablewidget4/5`.
- **Other modules used:**
  - *System library modules:*
    `chartlib/genbar1`, `chartlib/bltclass`, `chartlib/table_widget1`, `chartlib/test_format`, `chartlib/install_utils`, `lists`.

188.2 Documentation on exports (`table_widget4`)

tablewidget4/4: PREDICATE

`tablewidget4(Title,Header,ElementTable,Footer)`

Shows a regular table in a window. The system sets a default background image for the widget.

Example:

```
tablewidget4('Some sterEUtypes',
  'Source: Eurostat yearbook, 1999',
  [['Country','Adult alcohol intake per year (litres)',
       'Cigarettes smoked per day per adult',
            'Suicides per 100000 people'],
      ['Finland','8.4','2.2','26.3'],['Spain','11.4','5.3','7.5'],
      ['Austria','11.9','4.6','20.7'],['Britain','9.4','4.2','7.1'],
      ['USA','4.7','4.9','13'],['European Union','11.1','4.5','11.9']],
  'This is part of the published table').
```

Usage:

- *The following properties should hold at call time:*

  `Title` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

  `Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

  table_widget1:table(ElementTable) (table_widget1:table/1)

  `Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

tablewidget4/5: PREDICATE

`tablewidget4(Title,Header,ElementTable,Footer,BackgroundImage)`

This predicate and `tablewidget4/4` are comparable, except that in the already defined you must set the background image.

Example:

```
tablewidget4('Some sterEUtypes',
  'Source: Eurostat yearbook, 1999',
  [['Country','Adult alcohol intake per year (litres)',
       'Cigarettes smoked per day per adult',
           'Suicides per 100000 people'],
      ['Finland','8.4','2.2','26.3'],['Spain','11.4','5.3','7.5'],
      ['Austria','11.9','4.6','20.7'],['Britain','9.4','4.2','7.1'],
      ['USA','4.7','4.9','13'],['European Union','11.1','4.5','11.9']],
  'This is part of the published table',
  './images/rain.gif').
```

Usage:

– *The following properties should hold at call time:*

`Title` is a text (an atom) to be used as label, usually not very long. (genbar1:title/1)

`Header` is a text (an atom) describing the header of the graph. (genbar1:header/1)

table_widget1:table(ElementTable) (table_widget1:table/1)

`Footer` is a text (an atom) describing the footer of the graph. (genbar1:footer/1)

table_widget1:image(BackgroundImage) (table_widget1:image/1)

189 test_format (library)

Author(s): Isabel Martín García.

Most of the predicates exported by this module perform some checks to determine whether the arguments attain some conditions or not. In the second case an exception will be thrown. To catch the exceptions you can use the following metapredicates when invoking chartlib exported predicates:

- `chartlib_text_error_protect/1`
- `chartlib_text_error_protect/1`

Both metapredicates are defined in the chartlib_errhandle module that comes with this library. Some of the predicates have a `Predicate` argument which will be used in case of error to show which chartlib predicate causes the error.

189.1 Usage and interface (test_format)

- **Library usage:**
  `:- use_module(library(test_format)).`
- **Exports:**
  - *Predicates:*
    `equalnumber/3`, `not_empty/4`, `not_empty/3`, `check_sublist/4`, `valid_format/4`, `vectors_format/4`, `valid_vectors/4`, `valid_attributes/2`, `valid_table/2`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

189.2 Documentation on exports (test_format)

equalnumber/3: PREDICATE

`equalnumber(X,Y,Predicate)`

Test whether the list `X` and the list `Y` contain the same number of elements.

Usage:

- *The following properties should hold at call time:*
  `X` is a list. (basic_props:list/1)
  `Y` is a list. (basic_props:list/1)
  `Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

not_empty/4: PREDICATE

`not_empty(X,Y,Z,Predicate)`

Tests whether at least one the lists `X`, `Y` or `Z` are empty.

Usage:

– *The following properties should hold at call time:*

`X` is a list. (basic_props:list/1)

`Y` is a list. (basic_props:list/1)

`Z` is a list. (basic_props:list/1)

`Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

not_empty/3: PREDICATE

`not_empty(X,Y,Predicate)`

Tests whether the lists `X` or `Y` are empty.

Usage:

– *The following properties should hold at call time:*

`X` is a list. (basic_props:list/1)

`Y` is a list. (basic_props:list/1)

`Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

check_sublist/4: PREDICATE

`check_sublist(List,Number,Number,Predicate)`

Tests if the number of elements in each sublist of `List` is `Number1` or `Number2`.

Usage:

– *The following properties should hold at call time:*

`List` is a list. (basic_props:list/1)

`Number` is currently instantiated to an integer. (term_typing:integer/1)

`Number` is currently instantiated to an integer. (term_typing:integer/1)

`Predicate` is an atom. (basic_props:atm/1)

valid_format/4: PREDICATE

`valid_format(XVector,YVector,BarsAttributes,Predicate)`

Tests the following restrictions:

- The `XVector` number of elements is the same as each `YVector` sublist number of elements.
- The `YVector` length is equal to `BarsAttributes` length.

Usage:

– *The following properties should hold at call time:*

`XVector` is a list. (basic_props:list/1)

`YVector` is a list. (basic_props:list/1)

`BarsAttributes` is a list. (basic_props:list/1)

`Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

vectors_format/4: PREDICATE

`vectors_format(XVector,YVectors,LinesAttributes,Predicate)`

Tests the following conditions:

- `YVectors` list and `LinesAttributes` list have the same number of elements.
- `XVector` list and each `YVectors` element have the same number of elements.
- Each sublist of `LinesAttributes` is composed of 5, 3 or 1 elements.

Usage:

– *The following properties should hold at call time:*

`XVector` is a list. (basic_props:list/1)

`YVectors` is a list. (basic_props:list/1)

`LinesAttributes` is a list. (basic_props:list/1)

`Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

valid_vectors/4: PREDICATE

`valid_vectors(XVector,YVectors,LinesAttributes,Predicate)`

Tests the following conditions:

- `XVector` list, `YVectors` list and `LinesAttributes` list have the same number of elements.
- Each sublist of `LinesAttributes` is composed of 5, 3 or 1 element.

Usage:

– *The following properties should hold at call time:*

`XVector` is a list. (basic_props:list/1)

`YVectors` is a list. (basic_props:list/1)

`LinesAttributes` is a list. (basic_props:list/1)

`Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

valid_attributes/2: PREDICATE

`valid_attributes(BarsAttibuttes,Predicate)`

Check if each `BarsAttibuttes` element is a list composed of one or four elements.

Usage:

– *The following properties should hold at call time:*

`BarsAttibuttes` is a list. (basic_props:list/1)

`Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

valid_table/2: PREDICATE

`valid_table(ElementTable,Predicate)`

All of the `ElementTable` sublists have the same number of elements and are not empty.

Usage:

– *The following properties should hold at call time:*

`ElementTable` is a list. (basic_props:list/1)

`Predicate` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

190 Doubly linked lists

Author(s): David Trallero Mena.

This library implements "doubly linked" lists, in the sense that they can be traversed in both directions with good complexity. An index is used for referencing the current element in the list. This index can be modified by the *next* and *prev* predicates. The value of the current index can be obtained via the *top* predicate

190.1 Usage and interface (ddlist)

- **Library usage:**
  `:- use_module(library(ddlist)).`
- **Exports:**
  - *Predicates:*
    `null_ddlist/1`, `create_from_list/2`, `to_list/2`, `next/2`, `prev/2`, `insert/3`, `insert_top/3`, `insert_after/3`, `insert_begin/3`, `insert_end/3`, `delete/2`, `delete_top/2`, `delete_after/2`, `remove_all_elements/3`, `top/2`, `rewind/2`, `forward/2`, `length/2`, `length_next/2`, `length_prev/2`, `ddlist_member/2`.
  - *Regular Types:*
    `ddlist/1`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

190.2 Documentation on exports (ddlist)

null_ddlist/1: PREDICATE

Usage: `null_ddlist(A)`

- *Description:* `NullList` is an empty ddlist.
- *The following properties should hold at call time:*
  `A` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  `A` is a "doubly linked" list. (ddlist:ddlist/1)

create_from_list/2: PREDICATE

Usage: `create_from_list(List,DDList)`

- *Description:* Creates a doubly linked list from normal list `List`.
- *The following properties should hold at call time:*
  `List` is a list. (basic_props:list/1)
  `DDList` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  `DDList` is a "doubly linked" list. (ddlist:ddlist/1)

to_list/2: PREDICATE

Usage: `to_list(DDList,List)`

- *Description:* Converts from doubly linked list to list.
- *The following properties should hold at call time:*

  `DDList` is a "doubly linked" list. (ddlist:ddlist/1)

  `List` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `List` is a list. (basic_props:list/1)

next/2: PREDICATE

Usage: `next(OldList,NewList)`

- *Description:* `NewList` is `OldList` but index is set to the element following the current element of OldList. It satisfies `next(A,B), prev(B,A)`.
- *The following properties should hold at call time:*

  `OldList` is a "doubly linked" list. (ddlist:ddlist/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

prev/2: PREDICATE

Usage: `prev(OldList,NewList)`

- *Description:* `NewList` is `OldList` but index is set to the element before the current element of `OldList`.
- *The following properties should hold at call time:*

  `OldList` is a "doubly linked" list. (ddlist:ddlist/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

insert/3: PREDICATE

Usage: `insert(List,Element,NewList)`

- *Description:* `NewList` is like `List` but with `Element` inserted **before** the current index. It satisfies `insert(X , A , Xp) , delete(Xp , X)`.
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Element` is any term. (basic_props:term/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

insert_top/3: PREDICATE

Usage: `insert_top(List,Element,NewList)`

- *Description:* Put `Element` as new top of `NewList` and push the rest of elements after it. It satisfies top(NewList , element)
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Element` is any term. (basic_props:term/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

insert_after/3: PREDICATE

Usage: `insert_after(List,Element,NewList)`

- *Description:* `NewList` is like `List` but with `Element` inserted **after** the current index. It satisfies `insert_after(X, A, Xp), delete_after(Xp, X)`.
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Element` is any term. (basic_props:term/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

insert_begin/3: PREDICATE

Usage: `insert_begin(List,Element,NewList)`

- *Description:* `NewList` is like `List` with `Element` as first element.
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Element` is any term. (basic_props:term/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

insert_end/3: PREDICATE

Usage: `insert_end(List,Element,NewList)`

- *Description:* `NewList` is like `List` with `Element` as last element.
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Element` is any term. (basic_props:term/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

delete/2: PREDICATE

Usage: `delete(OldList,NewList)`

- *Description:* `NewList` does not have the previous element (top(prev)) of `OldList`.
- *The following properties should hold at call time:*

  `OldList` is a "doubly linked" list. (ddlist:ddlist/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

delete_top/2: PREDICATE

Usage: `delete_top(OldList,NewList)`

- *Description:* `NewList` does not have the current element (top) of `OldList`.
- *The following properties should hold at call time:*

  `OldList` is a "doubly linked" list. (ddlist:ddlist/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

delete_after/2: PREDICATE

Usage: `delete_after(OldList,NewList)`

- *Description:* `NewList` does not have next element to current element (top) of `OldList`.
- *The following properties should hold at call time:*

  `OldList` is a "doubly linked" list. (ddlist:ddlist/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

remove_all_elements/3: PREDICATE

Usage: `remove_all_elements(OldList,E,NewList)`

- *Description:* Remove all elements that unify with `E` from `OldList`. `NewList` is the result of this operation. The pointer is not modified unless there it is pointing at element that unifies with `E`.
- *The following properties should hold at call time:*

  `OldList` is a "doubly linked" list. (ddlist:ddlist/1)

  `E` is currently a term which is not a free variable. (term_typing:nonvar/1)
- *The following properties should hold upon exit:*

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

top/2: PREDICATE

Usage: `top(List,Element)`

- *Description:* `Element` is the element pointed by index.
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Element` is any term. (basic_props:term/1)

rewind/2: PREDICATE

Usage: `rewind(OldList,NewList)`

- *Description:* `NewList` is the `OldList` but index is set to 0.
- *The following properties should hold at call time:*

  `OldList` is a "doubly linked" list. (ddlist:ddlist/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

forward/2: PREDICATE

Usage: `forward(OldList,NewList)`

- *Description:* `NewList` is the `OldList` but index is set to lentgh of `NewList`.
- *The following properties should hold at call time:*

  `OldList` is a "doubly linked" list. (ddlist:ddlist/1)

  `NewList` is a "doubly linked" list. (ddlist:ddlist/1)

length/2: PREDICATE

Usage: `length(List,Len)`

- *Description:* `Len` is the length of the `List`
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Len` is an integer. (basic_props:int/1)

length_next/2: PREDICATE

Usage: `length_next(List,Len)`

- *Description:* `Len` is the length from the current index till the end.
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Len` is an integer. (basic_props:int/1)

length_prev/2: PREDICATE

Usage: `length_prev(List,Len)`

- *Description:* `Len` is the length from the beginning till the current index.
- *The following properties should hold at call time:*

  `List` is a "doubly linked" list. (ddlist:ddlist/1)

  `Len` is an integer. (basic_props:int/1)

ddlist/1: REGTYPE

Usage: `ddlist(X)`

- *Description:* `X` is a "doubly linked" list.

ddlist_member/2: PREDICATE

Usage: `ddlist_member(X,DDList)`

- *Description:* Success if `X` is member of `DDList`. `X` first unifies with elements of the forward list, i.e. from the top till the end, and later with elements from the top to the beginning.
- *The following properties should hold at call time:*

  `X` is any term. (basic_props:term/1)

  `DDList` is a "doubly linked" list. (ddlist:ddlist/1)

190.3 Other information (`ddlist`)

Two simple examples of the use of the ddlist library package follow.

190.3.1 Using insert_after

```
:- module( ddl1 , _ , [] ).

:- use_module(library(ddlist)).

main(A,B):-
        % L = []
        null_ddlist( L ),
        % L = [1]
        insert_after( L  , 1 , L1 ),
        % L = [1,2]
        insert_after( L1 , 2 , L2 ),
        % L = [1,3,2]
        insert_after( L2 , 3 , L3 ),
        % L = [1,3,2] => A = [1]
        top( L3 , A ),
        % L = [3,2]
        next( L3 , PL3 ),
        % L = [3,2] => A = [3]
        top( PL3 , B ).
```

190.3.2 More Complex example

```
:- module( ddl2 , _ , [] ).

:- use_module(library(ddlist)).

main(A,B):-
        % L = []
        null_ddlist( L ),
        % L = [1]
        insert_after( L  , 1 , L1 ),
        % L = [1,2]
        insert_after( L1 , 2 , L2 ),
        % L = [1,2]
        insert( L2 , 3 , L3 ),
        % L = [3,1,2]
        prev( L3 , PL3 ),
        % L = [],
        forward( PL3 , FOR ),
        % L = [2]
        prev( FOR , FOR1 ),
        % L = [2] => A = 2
        top( FOR1 , A ),
        % L = [1,2]
        prev( FOR1 , FOR2 ),
        % L = [2]
        delete_after( FOR2 , FOR3 ),
        % L = [3,2]
        prev( FOR3, FOR4 ),
```

```
% L = [3,2] => B = 3
top( FOR4 , B ).
```

Ciao DHT Implementation

Author(s): Arsen Kostenko.

General documentation pending

191 Top-level user interface to DHT

Author(s): Arsen Kostenko.

This module contains just a top-level interface to the utilities provided by the DHT system in Ciao. The 'philosophy' of the current approach is that most details are hidden behind simple read/write primitives and that the handler is exposed to the client for a concrete DHT. By doing things this way we expect to preserve the simplicity of DHT usage while providing the freedom of being able to switch between various instances of the Ciao DHT system

191.1 Usage and interface (dht_client)

- **Library usage:**
  `:- use_module(library(dht_client)).`
- **Exports:**
  - *Predicates:*
    `dht_connect/2`, `dht_connect/3`, `dht_disconnect/1`, `dht_consult_b/4`, `dht_consult_nb/4`, `dht_extract_b/4`, `dht_extract_nb/4`, `dht_store/4`, `dht_hash/3`.
- **Other modules used:**
  - *System library modules:*
    `sockets/sockets`, `dht/dht_misc`.

191.2 Documentation on exports (dht_client)

dht_connect/2: PREDICATE

Connect to the DHT specified by `Server` (IP address).

Usage: `dht_connect(Server,Connection)`

- *Description:* Perform a straightforward connection from the client-side of a DHT node. The information about DHT node to connect to is supplied as `Server`. It could equally be a DNS name or an IP address.
- *The following properties should hold at call time:*
  `Server` is an atom. (basic_props:atm/1)
  `Connection` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  `Server` is ground. (basic_props:gnd/1)
  `Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

dht_connect/3: PREDICATE

Connect to the DHT specified by `Server` (IP address).

Usage: `dht_connect(Server,Port,Connection)`

– *Description:* Perform a straightforward connection from the client-side of a DHT node. The information about DHT node to connect to is supplied as combination of `Server` and `Port`, if a non standart port is used for server-to-client communication. It could equally be a DNS name or an IP address.
– *The following properties should hold at call time:*

`Server` is an atom. (basic_props:atm/1)

`Port` is an integer. (basic_props:int/1)

`Connection` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Server` is an atom. (basic_props:atm/1)

`Port` is an integer. (basic_props:int/1)

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

dht_disconnect/1: PREDICATE

Disconnect from DHT, identified by special supplied connection.

Usage: `dht_disconnect(Connection)`

– *Description:* Issues 'end_of_file' token to the stream supplied as `Connection` and closes it without delay.
– *The following properties should hold at call time:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

dht_consult_b/4: PREDICATE

Look either for exact predicate or predicate matching the pattern in DHT.

Usage: `dht_consult_b(Conn,Key,Value,Resp)`

– *Description:* `dht_consult/4` performs a lookup in the DHT represented by `Conn` (see `dht_connect/2`) and searches for `Value` previously associated with the `Key`. `Value` may be partially instantiated in which case matching against the tuples stored in DHT is performed.
– *The following properties should hold at call time:*

`Conn` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Resp` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Conn` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Resp` is ground. (basic_props:gnd/1)

dht_consult_nb/4: PREDICATE

Usage: `dht_consult_nb(Connection,Key,Value,Response)`

– *Description:* `dht_consult/4` performs a lookup in the DHT represented by `Conn` (see `dht_connect/2`) and searches for `Value` previously associated with the `Key`. `Value` may be partially instantiated in which case matching against the tuples stored in DHT is performed.

– *The following properties should hold at call time:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Response` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Response` is ground. (basic_props:gnd/1)

dht_extract_b/4: PREDICATE

Extract from DHT an exact predicate of type `Key(Value)` or one that matches given pattern.

Usage: `dht_extract_b(Connection,Key,Value,Response)`

– *Description:* This predicate extracts information from the DHT connected to by `Connection`, that is stored under key `Key` if it matches the pattern supplied as `Value`

– *The following properties should hold at call time:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Response` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Response` is ground. (basic_props:gnd/1)

dht_extract_nb/4: PREDICATE

Extract from DHT an exact predicate of type `Key(Value)` or one that matches given pattern.

Usage: `dht_extract_nb(Connection,Key,Value,Response)`

– *Description:* This predicate extracts information from the DHT connected to by `Connection`, that is stored under key `Key` if it matches the pattern supplied as `Value`

– *The following properties should hold at call time:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Response` is ground. (basic_props:gnd/1)

dht_store/4: PREDICATE

Store data to DHT in form of `Key(Value)` predicate. No free variables are allowed in predicate.

Usage: `dht_store(Connection,Key,Value,Response)`

– *Description:* The value provided in `Value` is stored under a key given as `Key` inside the DHT mentioned as `Connection`

– *The following properties should hold at call time:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Response` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Key` is an atom. (basic_props:atm/1)

`Value` is any term. (basic_props:term/1)

`Response` is ground. (basic_props:gnd/1)

dht_hash/3: PREDICATE

Get value of hash function for a given term.

Usage 1: `dht_hash(Connection,Value,Hash)`

– *Description:* Get (in `Hash`) the hash of `Value` as determined by the DHT pointed to by `Connection`. Implemented mostly for testing purposes.

– *The following properties should hold at call time:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Value` is any term. (basic_props:term/1)

`Hash` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Value` is any term. (basic_props:term/1)

`Hash` is an integer. (basic_props:int/1)

Usage 2: `dht_hash(Connection,Value,Hash)`

– *Description:* Check whether `Hash` is equal to the hash of `Value` as determined by the DHT pointed to by `Connection`. Implemented mostly for testing purposes.

– *The following properties should hold at call time:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Value` is any term. (basic_props:term/1)

`Hash` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`Connection` is a predicate of type `dht_connection/2`, where first argument is of type `atm/1` and second of type `streams_basic:stream/1` (dht_client:dht_connection_type/1)

`Value` is any term. (basic_props:term/1)

`Hash` is an integer. (basic_props:int/1)

192 Top-level interface to a DHT server

Author(s): Arsen Kostenko.

192.1 Usage and interface (dht_server)

- **Library usage:**
  `:- use_module(library(dht_server)).`
- **Exports:**
  - *Predicates:*
    `dht_server/1`, `dht_prolog/1`.
- **Other modules used:**
  - *System library modules:*
    `sockets/sockets`, `system`, `format`, `dht/dht_config`, `dht/dht_logic_misc`, `dht/dht_s2s`, `dht/dht_s2c`.

192.2 Documentation on exports (dht_server)

dht_server/1: PREDICATE

`main/1`: start the DHT server. Here is a step-by-step bahavior:

1. set all values of `dht_config.pl` module to default.
2. modify those, for which command-line arguments were supplied.
3. modify server number separately, since it depends on two command-line arguments : `--server-id` and `--hash-power`.
4. output resulting values to terminal.
5. start client side communication by executing `dht_s2c:dht_s2c_mian/0`.
6. start server side communication by executing `dht_s2s:dht_s2s_mian/0`.

Usage: `dht_server(Arguments)`

- *The following properties should hold at call time:*
  `Arguments` is associated with simple list, that represents pairs of command line arguments (argument and its value). Possible values are: `--join-host`, `--hash-power`, `--server-id`, `--s2c-port`, `--s2c-threads`, `--s2s-port`, `--s2s-threads`. All of the arguments (except `--join-host`) accept integer values. In case of `--join-host` value of argument should be equal to IP/DNS address of host running a copy of DHT. (dht_server:dht_arguments_list/1)

dht_prolog/1: PREDICATE

Another style to launch the DHT server. All the parameters are passed in the list. Members of the list are of type `prameter_name(parameter_value)`, where `parameter_name` corresponds to name of the command line argument without two leading dashes (`--`) and with internal dash replaced by the underscore (`_`). All the values are of integer type, except the value of `join_host`, which should be an atom.

193 Server to client communication module

Author(s): Arsen Kostenko.

This module describes the server-2-client side behavior of any node. Since the behavior is not very different this module shares a number of fearures with `dht_s2c.pl`:

- extensive usage of the Ciao threading mechanism;
- interface made as simple as possible;
- `dht_server.pl` module is the only usage point;
- parameters are passed through the `dht_config.pl` module.

193.1 Usage and interface (`dht_s2c`)

- **Library usage:**
  `:- use_module(library(dht_s2c)).`
- **Exports:**
  - *Predicates:*
    `dht_s2c_main/0`.
- **Other modules used:**
  - *System library modules:*
    `sockets/sockets`, `concurrency/concurrency`, `dht/dht_config`, `dht/dht_misc`, `dht/dht_logic`, `dht/dht_logic_misc`.

193.2 Documentation on exports (`dht_s2c`)

dht_s2c_main/0: PREDICATE

Generally speaking, this predicate must perform some common tasks like:

- listen to a port for incoming connections from clients,
- and process requests from clients.

All the parameters needed are received from initial configuration, which is stored in `dht_config.pl` module.

194 Server to server communication module

Author(s): Arsen Kostenko.

This module describes server-2-server side behavior of any node. Since this includes various communication and inspection tasks that should be performed in parallel, this module makes extensive usage of Ciao threading mechanism. On the other hand, module must be as simple as possible in terms of usage. Therefore, only a single predicate `dht_s2s_main/0` is exposed to outer world and rest of complexity is hidden inside. This predicate is used from `dht_server.pl` module. All the parameters are passed through `dht_config.pl` module.

194.1 Usage and interface (`dht_s2s`)

- **Library usage:**
  `:- use_module(library(dht_s2s)).`
- **Exports:**
  - *Predicates:*
    `dht_s2s_main/0`.
- **Other modules used:**
  - *System library modules:*
    `sockets/sockets`, `system`, `concurrency/concurrency`, `dht/dht_config`, `dht/dht_misc`, `dht/dht_logic`.

194.2 Documentation on exports (`dht_s2s`)

dht_s2s_main/0: PREDICATE

Generally speaking, this predicate must perform some conventional tasks like:

- set local part of DHT to initial state,
- contact a successor node if there is any,
- and finally launch several threads that listen to streams, process incoming requests and inspect state of local finger-table.

All parameters (like listen port, number of threads, host to join, etc) are received from `dht_config.pl` module

195 DHT-related logics

Author(s): Arsen Kostenko.

This module implements a core DHT functionality, like finger-table manipulation and lookup search. Keep in mind that remote calls (known as remote predicate calls in Prolog) are extracted to a separate module as well as low-lever database handling. In turn, the logic module is utilized by higher-lever modules, like the server-2-server and server-2-client communication modules.

`Id` is treated as the value of the hash function used all over the DHT.

195.1 Usage and interface (`dht_logic`)

- **Library usage:**
  `:- use_module(library(dht_logic)).`
- **Exports:**
  - *Predicates:*
    `dht_init/1`, `dht_finger/2`, `dht_successor/1`, `dht_check_predecessor/1`, `dht_closest_preceding_finger/2`, `dht_find_predecessor/2`, `dht_find_successor/2`, `dht_join/1`, `dht_notify/1`, `dht_stabilize/0`, `dht_fix_fingers/0`, `dht_id_by_node/2`, `dht_find_and_consult_b/2`, `dht_consult_server_b/3`, `dht_find_and_consult_nb/2`, `dht_consult_server_nb/3`, `dht_find_and_extract_b/2`, `dht_extract_from_server_b/3`, `dht_find_and_extract_nb/2`, `dht_extract_from_server_nb/3`, `dht_find_and_store/2`, `dht_store_to_server/4`.
- **Other modules used:**
  - *System library modules:*
    `dht/dht_rpr`, `dht/dht_config`, `dht/dht_logic_misc`, `dht/dht_storage`, `dht/dht_routing`.

195.2 Documentation on exports (`dht_logic`)

dht_init/1: PREDICATE

Predicated performs the initialization of a single-node DHT.

Usage: `dht_init(OwnId)`

- *Description:* The predicate is intended to perform various initialization functions like finger_table creation, hostname and IP address retrieval. The value supplied in `OwnId` is treated as value of the hash function used all over the DHT.
- *The following properties should hold at call time:*
  `OwnId` is an integer. (basic_props:int/1)

dht_finger/2: PREDICATE

Get the node identifier (composition of identifier and IP-address) by using index in the finger table. Prolog synonym for `finger[k].node`.

Usage: `dht_finger(Idx,Finger)`

- *Description:* `dht_finger/2` implements the array looking up necessary to retrive information from the finger table, where first argument `Idx` if acting as array index and `Finger` as corresponding array value.

– *The following properties should hold at call time:*
`Idx` is an integer. (basic_props:int/1)
`Finger` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Idx` is an integer. (basic_props:int/1)
`Finger` is ground. (basic_props:gnd/1)

dht_successor/1: PREDICATE

Wrapper around `dht_finger/1`, where first argument is defaulted to '1'

Usage: `dht_successor(Successor)`

– *Description:* `dht_successor/1` a simple wrapper around `dht_finger/2`, that looks up the first row of so-called finger-table
– *The following properties should hold at call time:*
`Successor` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Successor` is ground. (basic_props:gnd/1)

dht_check_predecessor/1: PREDICATE

`dht_check_predecessor/1` predicate is called by external nodes when they run `dht_stabilize/0`. Since there is no other way to learn about a predecessor's failure, but checking explicitly once predecessor is required. that is why behavior of `dht_check_predecessor` is following:

1. give 'nil' as predecessor if it is stored in database that way, which happens if node is unaware of any other nodes on the ring,
2. otherwise, get current value of predecessor,
3. try calling predecessor with any arbitrary predicate in our case `dht_successor/1`,
4. if an exception is issued by remote call - reset predecessor to 'nil',
5. get value of predecessor once again and bind this value to the answer. We also check that in case of finger-nodes failure, but this would work out only in poorly populated Chord-ring

Usage 1: `dht_check_predecessor(PredecessorId)`

– *Description:* Get value of predecessor for current node.
– *The following properties should hold at call time:*
`PredecessorId` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`PredecessorId` is ground. (basic_props:gnd/1)

Usage 2: `dht_check_predecessor(PredecessorId)`

– *Description:* Check whether value supplied is equal to id of predecessor.
– *The following properties should hold at call time:*
`PredecessorId` is ground. (basic_props:gnd/1)
– *The following properties should hold upon exit:*
`PredecessorId` is ground. (basic_props:gnd/1)

dht_closest_preceding_finger/2: PREDICATE

Perform search over current finger table, in order to find entry pointing to node that is more closely located to value supplied via `Id`. Here is original part of code from Chord paper.

```
n.closest_preceding_finger(id){
    for i = m downto 1 {
       if (finger[i].node in (n, id)){
          return finger[i].node;
       }
    }
    return n;
}
```

.

Usage: `dht_closest_preceding_finger(Id,Finger)`

- *Description:* `dht_closest_preceding_finger/2` is searching local finger-table for the entry which points to a node, that is 'closer' (in terms of DHT distance) to the identifier specified at `Id`. It should hold that identifier of the node found is greater than identifier of current node and lesser than the identifier supplied in `Id`. No estimations concerning other nodes in between them are made. Furthermore, a cyclic structure of identifiers is preserved: identifier '0' is lesser then identifier '1' but greater then the highest identifier in DHT.
- *The following properties should hold at call time:*

  `Id` is an integer. (basic_props:int/1)

  `Finger` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `Id` is an integer. (basic_props:int/1)

  `Finger` is ground. (basic_props:gnd/1)

dht_find_predecessor/2: PREDICATE

Perform search over existing DHT structure. The resulting node is expected to precede the one identified by `Id`. Again, a small quote from Chord-paper source code.

```
n.find_predecessor(id){
    n' is n,
    while(id not_in (n', n'.successor] ){
       n' = n'.closest_preceding_finger(id);
    }
    return n';
}
```

Usage: `dht_find_predecessor(Id,Predecessor)`

- *Description:* `dht_find_predecessor/2` searches all over DHT for a node, which is 'the closest' (in terms of DHT distance) to the index specified as `Id`. In other words, the searched node must have the index lesser than the identifier specified in `Id` and there must not be any other nodes between identifier specified in `Id` and the node found.
- *The following properties should hold at call time:*

  `Id` is an integer. (basic_props:int/1)

  `Predecessor` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Id` is an integer. (basic_props:int/1)

`Predecessor` is ground. (basic_props:gnd/1)

dht_find_successor/2: PREDICATE

Usage: `dht_find_successor(Id,Successor)`

– *Description:* `dht_find_successor/2` has behavior similar to `dht_find_predecessor/2` except that it gives a node that directly succeeds the identifier specifies. In other words, the identifier of the node found is greater than the identifier specified in `Id` and there are no other nodes between the node found and the node specified in `Id`

– *The following properties should hold at call time:*

`Id` is an integer. (basic_props:int/1)

`Successor` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Id` is an integer. (basic_props:int/1)

`Successor` is ground. (basic_props:gnd/1)

dht_join/1: PREDICATE

It performs the join procedure on the node pointed by the argument. Here is the corresponding quote:

```
n.join(Next){
    predecessor = nil;
    successor = Next.find_successor(n);
}
```

Usage: `dht_join(NodeIP)`

– *Description:* `dht_join/2` is executed every time some node decides to join some DHT. The only information needed for execution of join is a valid ID / IP combination of any existing node, which must be supplied with `NodeId`

– *The following properties should hold at call time:*

`NodeIP` is an atom. (basic_props:atm/1)

dht_notify/1: PREDICATE

This predicate performs 'notification'. Notification is part of adaptation process launched by `dht_stabilize/0`. A quote from Chord paper that refers to this section is given below:

```
n.notify(NewNodeId){
    if (predecessor is nil ||
          NewNodeId in (predecessor, n)){
        predecessor = NewNodeId;
    }
}
```

Usage: `dht_notify(NewNode)`

– *Description:* `dht_notify/1` is called on current node, once any other DHT node specified by `NewNode` regards current node at it's successor.
– *The following properties should hold at call time:*
`NewNode` is ground. (basic_props:gnd/1)

dht_stabilize/0: PREDICATE

`dht_stabilize`

Perform stabilization on the successor node by asking successor's predecessor value. If the value returned is equal to current node, just stay calm, otherwise try to adopt the newly returned result as new successor. Most of the 'dirty' work is performed by `stabilize_successor/2` predicate. The corresponding Chord-paper quote is given below:

```
n.stabilize(){
    x = successor.predecessor;
    if (x in (n, successor)){
        successor = x;
    }
    successor.notify(n);
}
```

There is also a small simplification scheme, since before calling `dht_join/1` each node assumes itself as the only member of the circle, the very first case of `dht_stabilize/0` does nothing if successor is actually equal to current node.

Usage:

– *Description:* `dht_stabilize/0` is a eternally repeated predicate, which aims at stabilizing first-level references of the finger-table, while multiple concurrent joins and failures can happen. Its main goal is to ask its own successor to report its predecessor. If the reported node is different from the one that is calling `dht_stabilize/0`, there should be someone in between current node and its the successor. So a newly reported node should be registered as the successor instead of an old one. Technically, once an inconsistency between actual state of DHT and finger-table is found a newly found node is asked to perform `dht_notify/1`, with current node as an argument.

dht_fix_fingers/0: PREDICATE

`dht_fix_fingers`

This predicate takes care of maintaining the finger table entries (except the first one) up to date. The first entry is taken care of by a specialized predicate `dht_stabilize/0`, since there is much more responsibility on first entry (a direct successor). Rest of the same task is performed here, by `dht_fix_fingers/0`. In the Chord paper notation this part look like:

```
n.fix_fingers(){
    i = random index > 1 into finger[];
    finger[i].node = find_successor(finger[i].start);
}
```

.

Usage:

– *Description:* `dht_fix_fingers/0` is yet another randomly repeated predicate, which aims at stabilizing high-level references of finger-table, while multiple concurrent joins and failures happen. Once started, `dht_fix_fingers/0` picks a random (but not first) entry of finger table, and searches for a node responsible for the identifiers specified in the entry. In case, node that was found is different from the one currently specified, current node is replaced by a new one.

dht_id_by_node/2: PREDICATE

This predicate is entirely used by remote calls. Since there is `dht_rpr_id_by_node/2` for current module. This is an effort to avoid module-re-exportation (`dht_routing.pl` and `dht_rpr.pl`).

Usage: `dht_id_by_node(Node,NodeID)`

– *Description:* Get node identity by node number.

– *The following properties should hold at call time:*

`Node` is an integer. (basic_props:int/1)

`NodeID` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Node` is an integer. (basic_props:int/1)

`NodeID` is ground. (basic_props:gnd/1)

dht_find_and_consult_b/2: PREDICATE

`dht_find_and_consult_b/2` is meant to perform two actions:

- search for the node responsible for value supplied as a `Key`;
- perform `dht_consult_server_b/3` on a node found.

This is the first predicate of the whole `dht_find_and_*/2` family of predicates. Name of each member of the family gives a hint on implementation. And as it is presented by the name of the family, they share some part of implementation - all the members of the family perform search for particular DHT-node, which is achieved by `find_server/3` predicate.

Usage 1: `dht_find_and_consult_b(Key,Value)`

– *Description:* An invocation of this type would bind all free variables in `Value` to the values present in the local database of the node responsible for `Key`.

– *The following properties should hold at call time:*

`Key` is an integer. (basic_props:int/1)

`Value` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Key` is an integer. (basic_props:int/1)

`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_find_and_consult_b(Key,Value)`

– *Description:* Invocation with two ground arguments is equal to checking the presence of the term specified by `Value` inside the DHT.

– *The following properties should hold at call time:*

`Key` is an integer. (basic_props:int/1)

`Value` is ground. (basic_props:gnd/1)

dht_consult_server_b/3: PREDICATE

Basically forms the second half of `dht_find_and_consult/2` predicate. This predicate takes no care about search of corresponding server. The only thing it does - given a server try to locate a predicate under given `Key` there. Of course there two possible variants of application:

Usage 1: `dht_consult_server_b(NodeId,Key,Value)`

- *Description:* First variant deals with real consulting, when the value is unknown. Therefore, after predicate is successfully executed, a real value, that is stored under `Key` is associated with third argument (`Value`)
- *The following properties should hold at call time:*

  `NodeId` is ground. (basic_props:gnd/1)

  `Key` is ground. (basic_props:gnd/1)

  `Value` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `NodeId` is ground. (basic_props:gnd/1)

  `Key` is ground. (basic_props:gnd/1)

  `Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_consult_server_b(NodeId,Key,Value)`

- *Description:* Second variant is more similar to check for existence, All arguments are ground by the time predicate is called, so the only use is success/failure of predicate itself.
- *The following properties should hold at call time:*

  `NodeId` is ground. (basic_props:gnd/1)

  `Key` is ground. (basic_props:gnd/1)

  `Value` is ground. (basic_props:gnd/1)
- *The following properties should hold upon exit:*

  `NodeId` is ground. (basic_props:gnd/1)

  `Key` is ground. (basic_props:gnd/1)

  `Value` is ground. (basic_props:gnd/1)

dht_find_and_consult_nb/2: PREDICATE

`dht_find_and_consult_nb/2` is meant to perform two actions:

- search for the node responsible for value supplied as a `Key`;
- perform `dht_consult_server_nb/3` on a node found.

This is the first predicate of the whole `dht_find_and_*/2` family of predicates. Name of each member of the family gives a hint on implementation. And as it is presented by the name of the family, they share some part of implementation - all the members of the family perform search for particular DHT-node, which is achieved by `find_server/3` predicate.

Usage 1: `dht_find_and_consult_nb(Key,Value)`

- *Description:* An invocation of this type would bind all free variables in `Value` to the values present in the local database of the node responsible for `Key`.
- *The following properties should hold at call time:*

  `Key` is an integer. (basic_props:int/1)

  `Value` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Key` is an integer. (basic_props:int/1)

`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_find_and_consult_nb(Key,Value)`

– *Description:* Invocation with two ground arguments is equal to checking the presence of the term specified by `Value` inside the DHT.

– *The following properties should hold at call time:*

`Key` is an integer. (basic_props:int/1)

`Value` is ground. (basic_props:gnd/1)

dht_consult_server_nb/3: PREDICATE

Basically forms the second half of `dht_find_and_consult/2` predicate. This predicate takes no care about search of corresponding server. The only thing it does - given a server try to locate a predicate under given `Key` there. Of course there two possible variants of application:

Usage 1: `dht_consult_server_nb(NodeId,Key,Value)`

– *Description:* First variant deals with real consulting, when the value is unknown. Therefore, after predicate is successfully executed, a real value, that is stored under `Key` is associated with third argument (`Value`)

– *The following properties should hold at call time:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_consult_server_nb(NodeId,Key,Value)`

– *Description:* Second variant is more similar to check for existence, All arguments are ground by the time predicate is called, so the only use is success/failure of predicate itself.

– *The following properties should hold at call time:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is ground. (basic_props:gnd/1)

– *The following properties should hold upon exit:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is ground. (basic_props:gnd/1)

dht_find_and_extract_b/2: PREDICATE

`dht_find_and_extract_b/2` is meant to perform two actions:

- search for node responsible for value supplied as a `Key`;

- perform `dht_extract_from_server_b/3` on a node found.

Unlike `dht_find_and_consult/2`, this predicate removes matching records form the local databases of the corresponding nodes.

Usage 1: `dht_find_and_extract_b(Key,Value)`

– *Description:* An invocation of this type would bind all free variables in `Value` to values present in the local database of the corresponding node, and the values against which the matching was performed are erased.

– *The following properties should hold at call time:*

`Key` is an integer. (basic_props:int/1)

`Value` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Key` is an integer. (basic_props:int/1)

`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_find_and_extract_b(Key,Value)`

– *Description:* Invocation with two ground arguments is equal to checking presence of term specified by `Value` inside the DHT, and removing it in case of successful search.

– *The following properties should hold at call time:*

`Key` is an integer. (basic_props:int/1)

`Value` is ground. (basic_props:gnd/1)

dht_extract_from_server_b/3: PREDICATE

This predicate is also used entirely as a second part of `dht_find_and_extract/2` predicate. It's behavior differs from similar `dht_consult_server_b/3` predicate, only in action that is performed over DHT. In this case 'extraction' of predicates, instead of simple 'consultation'

Usage 1: `dht_extract_from_server_b(NodeId,Key,Value)`

– *Description:* First variant deals with blind 'extraction', when the value is unknown. Therefore, after predicate is successfully executed, a real value, that is stored under `Key` is associated with third argument (`Value`)

– *The following properties should hold at call time:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_extract_from_server_b(NodeId,Key,Value)`

– *Description:* Second variant is more similar to pointed elimination. All arguments are ground by the time predicate is called, so the only use is success and elimination of predicate itself or general failure on contrary.

– *The following properties should hold at call time:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is ground. (basic_props:gnd/1)

– *The following properties should hold upon exit:*
`NodeId` is ground. (basic_props:gnd/1)
`Key` is ground. (basic_props:gnd/1)
`Value` is ground. (basic_props:gnd/1)

dht_find_and_extract_nb/2: PREDICATE

`dht_find_and_extract_nb/2` is meant to perform two actions:

- search for node responsible for value supplied as a `Key`;
- perform `dht_extract_from_server_nb/3` on a node found.

Unlike `dht_find_and_consult/2`, this predicate removes matching records form the local databases of the corresponding nodes.

Usage 1: `dht_find_and_extract_nb(Key,Value)`

– *Description:* An invocation of this type would bind all free variables in `Value` to values present in the local database of the corresponding node, and the values against which the matching was performed are erased.
– *The following properties should hold at call time:*
`Key` is an integer. (basic_props:int/1)
`Value` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Key` is an integer. (basic_props:int/1)
`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_find_and_extract_nb(Key,Value)`

– *Description:* Invocation with two ground arguments is equal to checking presence of term specified by `Value` inside the DHT, and removing it in case of successful search.
– *The following properties should hold at call time:*
`Key` is an integer. (basic_props:int/1)
`Value` is ground. (basic_props:gnd/1)

dht_extract_from_server_nb/3: PREDICATE

This predicate is also used entirely as a second part of `dht_find_and_extract/2` predicate. It's behavior differs from similar `dht_consult_server_b/3` predicate, only in action that is performed over DHT. In this case 'extraction' of predicates, instead of simple 'consultation'

Usage 1: `dht_extract_from_server_nb(NodeId,Key,Value)`

– *Description:* First variant deals with blind 'extraction', when the value is unknown. Therefore, after predicate is successfully executed, a real value, that is stored under `Key` is associated with third argument (`Value`)
– *The following properties should hold at call time:*
`NodeId` is ground. (basic_props:gnd/1)
`Key` is ground. (basic_props:gnd/1)
`Value` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`NodeId` is ground. (basic_props:gnd/1)
`Key` is ground. (basic_props:gnd/1)
`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_extract_from_server_nb(NodeId,Key,Value)`

– *Description:* Second variant is more similar to pointed elimination. All arguments are ground by the time predicate is called, so the only use is success and elimination of predicate itself or general failure on contrary.

– *The following properties should hold at call time:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is ground. (basic_props:gnd/1)

– *The following properties should hold upon exit:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`Value` is ground. (basic_props:gnd/1)

dht_find_and_store/2: PREDICATE

First part of this predicate is similar to rest of `dht_find_and_*/2` family - perform search over existing DHT structure. After search - storage operation is performed.

Usage: `dht_find_and_store(Key,Value)`

– *Description:* `dht_find_and_store/2` is meant to perform two actions:

- search for a node responsible for the value supplied as `Key`;
- perform `dht_store_to_server/4` on node found.

Since the arguments of `dht_store_to_server/4` must be ground both arguments of `dht_find_and_store` are ground too.

– *The following properties should hold at call time:*

`Key` is an integer. (basic_props:int/1)

`Value` is ground. (basic_props:gnd/1)

dht_store_to_server/4: PREDICATE

Second part of `dht_find_and_store/2` predicate.

Usage: `dht_store_to_server(NodeId,Key,KeyHash,Value)`

– *Description:* Store `Value` under given `Key` and also record relation between `Key` and `KeyHash`.

– *The following properties should hold at call time:*

`NodeId` is ground. (basic_props:gnd/1)

`Key` is ground. (basic_props:gnd/1)

`KeyHash` is ground. (basic_props:gnd/1)

`Value` is ground. (basic_props:gnd/1)

196 Finger table and routing information

Author(s): Arsen Kostenko.

This module holds the basic operations over the finger table and other routing information. Developed for the sake of simplicity, it should be used only within the `dht_logic.pl` file. Probably the most important and abstract part of the routing is the finger table. Here is a finger table copy of the node representing identifier 0 (from the Chord paper):

```
finger_table(1, 1, finger_interval(1,2), 1).
 finger_table(2, 2, finger_interval(2,4), 3).
 finger_table(3, 4, finger_interval(4,0), 0).
```

.

This table reflects the case where there are 8 identifiers in total in the identifier circle and three running nodes with identifiers 0, 1, and 3. In general each entry can be represented as `finger_table(idx, start, interval, succ)`. Here is a brief description of each argument:

- `idx` - Since arrays are not native for Prolog, I put the index as the first argument to the finger_table predicate to achieve indexing.
- `start` - ID that starts the corresponding section of the table.
- `interval` - IDs that fit into the corresponding section of table, represented in the form: `finger_interval(interval_start, interval_end).`, where:
  - interval_start - ID that starts this interval.
  - interval_end - first ID of the next interval.
- `succ` - ID of node responsible for the corresponding section of table, represented in `node_id(id, ip).` form, where:
  - id - ID of a node.
  - ip - IP address of a node.

196.1 Usage and interface (`dht_routing`)

- **Library usage:**
  `:- use_module(library(dht_routing)).`
- **Exports:**
  - *Predicates:*
    `dht_finger_table/2`, `dht_finger_start/2`, `dht_update_finger/2`, `dht_set_finger/4`, `dht_predecessor/1`, `dht_set_predecessor/1`, `dht_reset_predecessor/0`.

196.2 Documentation on exports (`dht_routing`)

dht_finger_table/2: PREDICATE

It serves as a read-only interface to the finger table information as follows: `Idx` stands for an index in an array, and `Node` is the number of the node which is recorded in the entry with that index.

Usage 1: `dht_finger_table(Idx,Node)`

- *Description:* This case is equal to getting any (possibly random) entry from the finger table.

– *The following properties should hold at call time:*

`Idx` is a free variable. (term_typing:var/1)

`Node` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Idx` is an integer. (basic_props:int/1)

`Node` is an integer. (basic_props:int/1)

Usage 2: `dht_finger_table(Idx,Node)`

– *Description:* Get some (one or several if backtracking is exploited) indexes that mention particular node numer, supplied as `Node`.

– *The following properties should hold at call time:*

`Idx` is a free variable. (term_typing:var/1)

`Node` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`Idx` is an integer. (basic_props:int/1)

`Node` is an integer. (basic_props:int/1)

Usage 3: `dht_finger_table(Idx,Node)`

– *Description:* Get the identifier of the node, indexed by `Idx`. Theoretically, there should be no more than on entry for each value of index.

– *The following properties should hold at call time:*

`Idx` is an integer. (basic_props:int/1)

`Node` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Idx` is an integer. (basic_props:int/1)

`Node` is an integer. (basic_props:int/1)

Usage 4: `dht_finger_table(Idx,Node)`

– *Description:* Check whether a particular combination (index and node number) is present in finger table.

– *The following properties should hold at call time:*

`Idx` is an integer. (basic_props:int/1)

`Node` is an integer. (basic_props:int/1)

dht_finger_start/2: PREDICATE

This predicate may be regarded as a read-only interface to map between the index is a finger table and the beginning of the section of the DHT circle.

Usage 1: `dht_finger_start(Idx,NextId)`

– *Description:* Get any entry, in other words and entry that has any index and points to any segment in the circle. Common constraints on finger tables, do hold anyway:

- result unifies with the content of finger table;
- predicate enumerates possible results on backtracking.

– *The following properties should hold at call time:*

`Idx` is a free variable. (term_typing:var/1)

`NextId` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*
`Idx` is an integer. (basic_props:int/1)
`NextId` is an integer. (basic_props:int/1)

Usage 2: `dht_finger_start(Idx,NextId)`
– *Description:* Ask for the index of entry that mentions a particular node as the starting one.
– *The following properties should hold at call time:*
`Idx` is a free variable. (term_typing:var/1)
`NextId` is an integer. (basic_props:int/1)
– *The following properties should hold upon exit:*
`Idx` is an integer. (basic_props:int/1)
`NextId` is an integer. (basic_props:int/1)

Usage 3: `dht_finger_start(Idx,NextId)`
– *Description:* Get the starting node for a certain entry of finger table.
– *The following properties should hold at call time:*
`Idx` is an integer. (basic_props:int/1)
`NextId` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Idx` is an integer. (basic_props:int/1)
`NextId` is an integer. (basic_props:int/1)

Usage 4: `dht_finger_start(Idx,NextId)`
– *Description:* Check for the presence of a correspondence between a certain entry and its starting node.
– *The following properties should hold at call time:*
`Idx` is an integer. (basic_props:int/1)
`NextId` is an integer. (basic_props:int/1)

dht_update_finger/2: PREDICATE

`dht_update_finger/2` takes care of changing node value to `NodeId` for entry with index number equal to `Idx`. This predicate has entirely imperative behavior. Both arguments are to be ground at the moment of call.

Usage: `dht_update_finger(Idx,NodeId)`
– *Description:* Stores supplied information, previously erasing all old information.
– *The following properties should hold at call time:*
`Idx` is an integer. (basic_props:int/1)
`NodeId` is an integer. (basic_props:int/1)

dht_set_finger/4: PREDICATE

This predicate sets the value of finger table entries. If any entry with the same index (`Idx`) exists, it is erased before new value is asserted.

Usage: `dht_set_finger(Idx,Start,End,Node)`
– *Description:* All the arguments are to be ground and of type integer.

– *The following properties should hold at call time:*
`Idx` is an integer. (basic_props:int/1)
`Start` is an integer. (basic_props:int/1)
`End` is an integer. (basic_props:int/1)
`Node` is an integer. (basic_props:int/1)

dht_predecessor/1: PREDICATE

A common read-only interface to get or check the ID of the predecessor node.

Usage 1: `dht_predecessor(PredId)`

– *Description:* Get number of predecessor node.
– *The following properties should hold at call time:*
`PredId` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`PredId` is an integer. (basic_props:int/1)

Usage 2: `dht_predecessor(PredId)`

– *Description:* Check whether particular number is associated with current predecessor.
– *The following properties should hold at call time:*
`PredId` is an integer. (basic_props:int/1)

dht_set_predecessor/1: PREDICATE

Write interface for predecessor ID. It checks the type of its argument and saves argument locally.

Usage: `dht_set_predecessor(PredId)`

– *Description:* Save `PredId` as the ID of predecessor for current node.
– *The following properties should hold at call time:*
`PredId` is an integer. (basic_props:int/1)

dht_reset_predecessor/0: PREDICATE

Set value of preceding node index to 'nil' whatever it's prior value is.

197 Various wrappers for DHT logics module

Author(s): Arsen Kostenko.

This module contains miscellaneous predicates related to the `dht_logic.pl` module. Mostly various calculation-wrappers.

197.1 Usage and interface (`dht_logic_misc`)

- **Library usage:**
  `:- use_module(library(dht_logic_misc)).`
- **Exports:**
  - *Predicates:*
    `hash_size/1`, `highest_hash_number/1`, `consistent_hash/2`, `next_on_circle/2`, `not_in_circle_oc/3`, `in_circle_oo/3`, `in_circle_oc/3`.
- **Other modules used:**
  - *System library modules:*
    `indexer/hash`, `dht/dht_config`.

197.2 Documentation on exports (`dht_logic_misc`)

hash_size/1: PREDICATE

This predicate calculates `(2**m)`, where 'm' is equal to `hash_power/1` received from system wide configurations stored in `dht_config.pl`.

Usage: `hash_size(Size)`

- *Description:* The only purpose of `hash_size/1` predicate is to get corresponding value from system wide configurations stored in `dht_config.pl` file and convert it into the number of nodes virtually available in the current DHT installation.
- *The following properties should hold at call time:*
  `Size` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  `Size` is ground. (basic_props:gnd/1)
  `Size` is an integer. (basic_props:int/1)

highest_hash_number/1: PREDICATE

This predicate calculates `(2**m)-1`, where 'm' is equal to `hash_power/1` received from system wide configurations stored in `dht_config.pl`.

Usage 1: `highest_hash_number(N)`

- *Description:* If used with a ground argument `highest_hash_number/1` simply checks whether the value supplied corresponds to the biggest hash number possible in the DHT.
- *The following properties should hold at call time:*
  `N` is an integer. (basic_props:int/1)

Usage 2: `highest_hash_number(N)`

- *Description:* Otherwise (if argument happens to be a free variable) the value of `N` is bound to the highest hash number possible in the DHT.
- *The following properties should hold at call time:*

  `N` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `N` is an integer. (basic_props:int/1)

consistent_hash/2: PREDICATE

This kind of computation is usually performed every time, when one needs to get value of hash-function.

Usage: `consistent_hash(Term,DHTHash)`

- *Description:* Straightforward computation of hash (`DHTHash`) on the basis of `Term` supplied.
- *The following properties should hold at call time:*

  `Term` is any term. (basic_props:term/1)

  `DHTHash` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `Term` is any term. (basic_props:term/1)

  `DHTHash` is an integer. (basic_props:int/1)

next_on_circle/2: PREDICATE

Performs the calculation of `(id+1) mod ((2**m)-1)`, where `id` is the first argument supplied, and `m` is equal to `hash_power/1`.

Usage: `next_on_circle(Id,Num)`

- *Description:* `next_on_circle/2` is a calculation wrapping. One can only use it with first argument ground and equal to integer value. What is more, that value **MUST** be kept within certain bounds: value of `Id` **MUST** meet the constraints `0<=Id<=HighNum`, where `HighNum` is retrieved via `highest_hash_number/1` predicate.
- *The following properties should hold at call time:*

  `Id` is an integer. (basic_props:int/1)

  `Num` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `Id` is an integer. (basic_props:int/1)

  `Num` is an integer. (basic_props:int/1)

not_in_circle_oc/3: PREDICATE

`not_in_circle_oc/3`: the 'oc' suffix stands for "open-closed" interval. The predicate implements the behavior of the expression: `Id not_in (a,b]`. where both 'a' and 'b' are numbers/identifiers on the DHT-ring.

Usage: `not_in_circle_oc(Id,Start,End)`

– *Description:* Yet another wrapper around simple calculations. All arguments are expected to be ground, by the time the predicate is called. The predicate checks whether the value supplied as `Id` fits into circle sector defined by values of `Start` and `End`. If the condition is not true - the whole predicate fails.
– *The following properties should hold at call time:*

`Id` is an integer. (basic_props:int/1)

`Start` is an integer. (basic_props:int/1)

`End` is an integer. (basic_props:int/1)

in_circle_oo/3: PREDICATE

`in_circle_oo/3`: the 'oo' suffix stands for "open-open" circle interval. The predicate implements the behavior of following expression: `Id in (a,b)`.

Usage: `in_circle_oo(Id,Start,End)`

– *Description:* Check whether value of `Id` fits into sector defined by `Start` and `End` excluding both, fail otherwise.
– *The following properties should hold at call time:*

`Id` is an integer. (basic_props:int/1)

`Start` is an integer. (basic_props:int/1)

`End` is an integer. (basic_props:int/1)

in_circle_oc/3: PREDICATE

`in_circle_oc/3`: the 'oc' suffix stands for "open-closed" interval. The predicate is based on `in_circle_oo/3` and considers the interval to be closed on the right. This predicate implements the behavior of the expression: `Id in (a,b]`

Usage: `in_circle_oc(Id,Start,End)`

– *Description:* Check whether value of `Id` fits into sector defined by `Start` and `End` excluding first and including second. Fail if the condition does not hold.
– *The following properties should hold at call time:*

`Id` is an integer. (basic_props:int/1)

`Start` is an integer. (basic_props:int/1)

`End` is an integer. (basic_props:int/1)

198 Remote predicate calling utilities

Author(s): Arsen Kostenko.

RPR stands for Remote PRedicate calling. The basic functionality of a remote procedure (or predicate, in Prolog terms of Prolog) call is gathered here. Thus module contains predicates for the execution of remote calls and manipulation routines of the module-specific database.

198.1 Usage and interface (dht_rpr)

- **Library usage:**
  `:- use_module(library(dht_rpr)).`
- **Exports:**
  - *Predicates:*
    `dht_rpr_register_node/1`, `dht_rpr_register_node/2`, `dht_rpr_node_by_id/2`, `dht_rpr_id_by_node/2`, `dht_rpr_compose_id/3`, `dht_rpr_clear_by_node/1`, `dht_rpr_node/1`, `dht_rpr_call/2`, `dht_rpr_call/3`, `node_id/2`.
  - *Regular Types:*
    `dht_rpr_node_id/1`.
- **Other modules used:**
  - *System library modules:*
    `format`, `sockets/sockets`, `dht/dht_config`, `dht/dht_misc`.

198.2 Documentation on exports (dht_rpr)

dht_rpr_register_node/1: PREDICATE

Sometimes information from remote nodes is received in the form of a `node_id/2`. In these cases a single argument predicate might be useful.

Usage: `dht_rpr_register_node(NodeID)`

- *Description:* Save node identity to module-specific DB.
- *The following properties should hold at call time:*
  `NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

dht_rpr_register_node/2: PREDICATE

General write interface to DB of physical nodes (those that have IP addresses). Information is re-written every time, so no old entries are expected to remain.

Usage: `dht_rpr_register_node(Node,NodeIP)`

- *Description:* `dht_rpr_register_node/2` is responsible for the management of a module-specific database that stores information on node identifiers and IP addresses corresponding to them. Despite there is only one usage mode the behavior may differ depending on the state of module-specific database. For instance, if the database already contains information about a node, whose identifier is equal to the one supplied in `Node`, a newly supplied entry would be written over the old one. A new entry is added to the module-specific database otherwise.

– *The following properties should hold at call time:*

`Node` is an integer. (basic_props:int/1)

`NodeIP` is ground. (basic_props:gnd/1)

dht_rpr_node_by_id/2: PREDICATE

Just a convenient wrapper around module-specific data structures, which is used in combination with Ciao functional syntax.

Usage 1: `dht_rpr_node_by_id(NodeID,Node)`

– *Description:* Extract value of node from the node identifier `NodeID`.

– *The following properties should hold at call time:*

`NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

`Node` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

`Node` is ground. (basic_props:gnd/1)

Usage 2: `dht_rpr_node_by_id(NodeID,Node)`

– *Description:* Check whether the value of the `NodeID` node identifier corresponds to the `Node` value.

– *The following properties should hold at call time:*

`NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

`Node` is ground. (basic_props:gnd/1)

– *The following properties should hold upon exit:*

`NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

`Node` is ground. (basic_props:gnd/1)

Usage 3: `dht_rpr_node_by_id(NodeID,Node)`

– *Description:* Return the node identifier associated to the exact node number.

– *The following properties should hold at call time:*

`NodeID` is a free variable. (term_typing:var/1)

`Node` is ground. (basic_props:gnd/1)

– *The following properties should hold upon exit:*

`NodeID` is a term of type `node_id/2` with first argument as integer and second as free variable or IP/DNS address (dht_rpr:dht_rpr_comp_node_addr/1)

`Node` is ground. (basic_props:gnd/1)

Usage 4: `dht_rpr_node_by_id(NodeID,Node)`

– *Description:* This case is merely useless, however perfectly possible.

– *The following properties should hold at call time:*

`NodeID` is a free variable. (term_typing:var/1)

`Node` is a free variable. (term_typing:var/1)

dht_rpr_id_by_node/2: PREDICATE

Convenient wrapper around internal data structures, as well as `dht_rpr_node_by_id/2`, this predicate is usually used in combination with Ciao functional syntax.

Usage 1: `dht_rpr_id_by_node(Node,NodeID)`

- *Description:* Get node identity by it's number.
- *The following properties should hold at call time:*

  `Node` is an integer. (basic_props:int/1)

  `NodeID` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `Node` is an integer. (basic_props:int/1)

  `NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

Usage 2: `dht_rpr_id_by_node(Node,NodeID)`

- *Description:* Lookup and fill ramaining id field.
- *The following properties should hold at call time:*

  `Node` is an integer. (basic_props:int/1)

  `NodeID` is a term of type `node_id/2` with first argument as free variable or integer and second an IP/DNS address (dht_rpr:dht_rpr_comp_node_id/1)
- *The following properties should hold upon exit:*

  `Node` is an integer. (basic_props:int/1)

  `NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

Usage 3: `dht_rpr_id_by_node(Node,NodeID)`

- *Description:* Lookup and fill remainting address field.
- *The following properties should hold at call time:*

  `Node` is an integer. (basic_props:int/1)

  `NodeID` is a term of type `node_id/2` with first argument as integer and second as free variable or IP/DNS address (dht_rpr:dht_rpr_comp_node_addr/1)
- *The following properties should hold upon exit:*

  `Node` is an integer. (basic_props:int/1)

  `NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

Usage 4: `dht_rpr_id_by_node(Node,NodeID)`

- *Description:* Check whether local database really has record about `NodeID` with `Node` as its node number.
- *The following properties should hold at call time:*

  `Node` is an integer. (basic_props:int/1)

  `NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

dht_rpr_node_id/1: REGTYPE

Usage: `dht_rpr_node_id(NodeID)`

- *Description:* `NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address.

dht_rpr_compose_id/3: PREDICATE

Compose identity structure out of arguments provided

Usage 1: `dht_rpr_compose_id(Node,NodeIP,NodeID)`

– *Description:* Compose an internal structure.

– *The following properties should hold at call time:*

`Node` is an integer. (basic_props:int/1)

`NodeIP` is ground. (basic_props:gnd/1)

`NodeID` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Node` is an integer. (basic_props:int/1)

`NodeIP` is ground. (basic_props:gnd/1)

`NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

Usage 2: `dht_rpr_compose_id(Node,NodeIP,NodeID)`

– *Description:* Check whether arguments correspond to structure supplied.

– *The following properties should hold at call time:*

`Node` is an integer. (basic_props:int/1)

`NodeIP` is ground. (basic_props:gnd/1)

`NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

– *The following properties should hold upon exit:*

`Node` is an integer. (basic_props:int/1)

`NodeIP` is ground. (basic_props:gnd/1)

`NodeID` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

dht_rpr_clear_by_node/1: PREDICATE

Write-interface (or more precisely, erase-interface) to DB of physical nodes (those that have IP-address). `dht_rpr_clear_node` is a dumb-wrapper around retraction operation over module-specific database. As usually, retraction may be performed when the argument is a free variable.

Usage 1: `dht_rpr_clear_by_node(Node)`

– *Description:* Erase any (possibly random) node information.

– *The following properties should hold at call time:*

`Node` is a term of type `node_id/2` with first argument as integer and second as free variable or IP/DNS address (dht_rpr:dht_rpr_comp_node_addr/1)

– *The following properties should hold upon exit:*

`Node` is ground. (basic_props:gnd/1)

Usage 2: `dht_rpr_clear_by_node(Node)`

– *Description:* Erase exact node.

– *The following properties should hold at call time:*

`Node` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

– *The following properties should hold upon exit:*

`Node` is ground. (basic_props:gnd/1)

dht_rpr_node/1: PREDICATE

Generic read interface to DB of physical nodes.

Usage 1: `dht_rpr_node(Node)`

– *Description:* Checks for presence of any information on a node supplied as `Node`.

– *The following properties should hold at call time:*

`Node` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

Usage 2: `dht_rpr_node(Node)`

– *Description:* Get node that matches given template.

– *The following properties should hold at call time:*

`Node` is a term of type `node_id/2` with first argument is either a free variable or an integer and second is either a free variable or a IP/DNS address (dht_rpr:dht_rpr_comp_node/1)

dht_rpr_call/2: PREDICATE

`dht_rpr_call/2` execute a goal remotely. The platform for remote execution is specified by `HostId`. `Goal` might be a fully instantiated term as well as partially instantiated one (as used in any other goal).

Usage 1: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of partially instantiated goal: e.g., a pattern search. Remote host for execution is specified via IP/DNS address of node identity, rest of identity is ignored

– *The following properties should hold at call time:*

`HostId` is a term of type `node_id/2` with first argument as free variable or integer and second an IP/DNS address (dht_rpr:dht_rpr_comp_node_id/1)

`Goal` is a free variable. (term_typing:var/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is a term of type `node_id/2` with first argument as free variable or integer and second an IP/DNS address (dht_rpr:dht_rpr_comp_node_id/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 2: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of partially instantiated goal: e.g., a pattern search. Remote host for execution is specified via integer of node identity, IP/DNS address is searched through local database.

– *The following properties should hold at call time:*

`HostId` is a term of type `node_id/2` with first argument as integer and second as free variable or IP/DNS address (dht_rpr:dht_rpr_comp_node_addr/1)

`Goal` is a free variable. (term_typing:var/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is a term of type `node_id/2` with first argument as integer and second as free variable or IP/DNS address (dht_rpr:dht_rpr_comp_node_addr/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 3: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of partially instantiated goal: e.g., a pattern search. Remote host for execution is specified via IP/DNS address of node identity, rest of identity is ignored

– *The following properties should hold at call time:*

`HostId` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

`Goal` is a free variable. (term_typing:var/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 4: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of partially instantiated goal: e.g., a pattern search.Remote host for execution is specified via integer, IP/DNS address is searched through local database using that integer as part of node identity.

– *The following properties should hold at call time:*

`HostId` is an integer. (basic_props:int/1)

`Goal` is a free variable. (term_typing:var/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is an integer. (basic_props:int/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 5: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of partially instantiated goal: e.g., a pattern search. Finally make a try to use first argument as directly-specified IP/DNS address.

– *The following properties should hold at call time:*

`HostId` is an atom. (basic_props:atm/1)

`Goal` is a free variable. (term_typing:var/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is an atom. (basic_props:atm/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 6: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of fully instantiated goal: e.g., a pattern search. Remote host for execution is specified via IP/DNS address of node identity, rest of identity is ignored

– *The following properties should hold at call time:*

`HostId` is a term of type `node_id/2` with first argument as free variable or integer and second an IP/DNS address (dht_rpr:dht_rpr_comp_node_id/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is a term of type `node_id/2` with first argument as free variable or integer and second an IP/DNS address (dht_rpr:dht_rpr_comp_node_id/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 7: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of fully instantiated goal: e.g., a pattern search. Remote host for execution is specified via integer of node identity, IP/DNS address is searched through local database.

– *The following properties should hold at call time:*

`HostId` is a term of type `node_id/2` with first argument as integer and second as free variable or IP/DNS address (dht_rpr:dht_rpr_comp_node_addr/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is a term of type `node_id/2` with first argument as integer and second as free variable or IP/DNS address (dht_rpr:dht_rpr_comp_node_addr/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 8: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of fully instantiated goal: e.g., a pattern search. Remote host for execution is specified via IP/DNS address of node identity, rest of identity is ignored

– *The following properties should hold at call time:*

`HostId` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is a term of type `node_id/2` with first argument as integer value and second as IP/DNS address. (dht_rpr:dht_rpr_node_id/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 9: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of fully instantiated goal: e.g., a pattern search.Remote host for execution is specified via integer, IP/DNS address is searched through local database using that integer as part of node identity.

– *The following properties should hold at call time:*

`HostId` is an integer. (basic_props:int/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is an integer. (basic_props:int/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

Usage 10: `dht_rpr_call(HostId,Goal)`

– *Description:* Perform remote call of fully instantiated goal: e.g., a pattern search. Finally make a try to use first argument as directly-specified IP/DNS address.

– *The following properties should hold at call time:*

`HostId` is an atom. (basic_props:atm/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

– *The following properties should hold upon exit:*

`HostId` is an atom. (basic_props:atm/1)

`Goal` is ground. (basic_props:gnd/1)

`Goal` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

dht_rpr_call/3: PREDICATE

The only difference this predicate has with `dht_rpr_call/2` is debug-enabling switch. Third parameter is expected to be responsible for that. It takes one of two possible values 'debug' or 'nodebug'. The first one is default. If default value is used, the standard output stream is populated with various debugging information.

Usage 1: `dht_rpr_call(HostId,Goal,Debug)`

– *Description:* `dht_rpr_call/3` executes a `Goal` remotely and prints some debugging information locally. The platform for the remote execution is specified by `HostId.Debug` variable must be bound to value 'debug' in order for this case to fire.

– *The following properties should hold at call time:*

`HostId` is an integer. (basic_props:int/1)

`Goal` is any term. (basic_props:term/1)

`Debug` is ground. (basic_props:gnd/1)

Usage 2: `dht_rpr_call(HostId,Term,Anything)`

– *Description:* `dht_rpr_call/3` executes a `Goal` remotely without writing any information on the local output stream. The platform for the remote execution is specified by `HostId`.

– *The following properties should hold at call time:*

`HostId` is an integer. (basic_props:int/1)

`Term` is any term. (basic_props:term/1)

`Anything` is a free variable. (term_typing:var/1)

node_id/2: PREDICATE

`node_id/2`is an auxilary prediate for internal data structures hanling.

The predicate is of type *concurrent*.

199 Underlying data-storage module

Author(s): Arsen Kostenko.

This module contains very low-level utilities of data storage specific to a single node. Neither data sharing nor remote invocation is performed at this level. Note that no dedicated data manipulation is performed neither on this level. Also keep in mind that concept of `Key` should always meet the same constraints as the second argument of `functor/3` predicate. Important decision taken at this point is representation of all information stored in the DHT as usual Prolog facts. This behavior is implemented by representing relations of form `Key -> Value` in form of `Key(Value)` facts.

199.1 Usage and interface (dht_storage)

- **Library usage:**
  `:- use_module(library(dht_storage)).`
- **Exports:**
  - *Predicates:*
    `dht_store/3`, `dht_extract_b/2`, `dht_extract_nb/2`, `dht_consult_b/2`, `dht_consult_nb/2`, `dht_key_hash/2`.
- **Other modules used:**
  - *System library modules:*
    `concurrency/concurrency`.

199.2 Documentation on exports (dht_storage)

dht_store/3: PREDICATE

This predicate stores information into the local (module-local) database. The three arguments stand for key, value, and value of hash-function, when applied to the key. The third argument is needed for auxiliary database, which is used by reverse lookup.

Usage: `dht_store(Key,KeyHash,Value)`

- *Description:* The value provided in `Value` is stored under a key given as `Key`.
- *The following properties should hold at call time:*

`Key` is an atom.	(basic_props:atm/1)
`KeyHash` is an integer.	(basic_props:int/1)
`Value` is ground.	(basic_props:gnd/1)

dht_extract_b/2: PREDICATE

Looks up local database for fact of following type `Key/1`, that takes `Value` as argument. The pair is extracted from the database if there is at lease one match. Otherwise call is blocking once the fact is not found instantly.

Usage 1: `dht_extract_b(Key,Value)`

- *Description:* Search for exact fact.

– *The following properties should hold at call time:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)
– *The following properties should hold upon exit:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_extract_b(Key,Value)`
– *Description:* Search for fact matching pattern. Pattern is supplied as `Value`.
– *The following properties should hold at call time:*
`Key` is an atom. (basic_props:atm/1)
`Value` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)

dht_extract_nb/2: PREDICATE

Looks up the local database for a fact of type `Key`/1 that takes `Value` as argument. If a corresponding fact is found in database, it is extracted from it. With respect to behavior of `dht_extract_b/2`, this predicated has a useful difference - it does not block while searching.

Usage 1: `dht_extract_nb(Key,Value)`
– *Description:* Check that combination of `Key`/`Value` appears in database.
– *The following properties should hold at call time:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)
– *The following properties should hold upon exit:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_extract_nb(Key,Value)`
– *Description:* Search for combination of `Key`/`Value` matching pattern. Pattern is supplied as `Value`.
– *The following properties should hold at call time:*
`Key` is an atom. (basic_props:atm/1)
`Value` is a free variable. (term_typing:var/1)
– *The following properties should hold upon exit:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)

dht_consult_b/2: PREDICATE

Looks up local database for combination of `Key`/`Value` of following type `Key`/1 that takes `Value` as argument. The combination of `Key`/`Value` is only looked up. If there is no corresponding combination of `Key`/`Value` in local database, `dht_consult_b/2` blocks until such fact is inserted. Nothing is done neither to combination of `Key`/`Value` nor to local database.

Usage 1: `dht_consult_b(Key,Value)`

– *Description:* Search for exact combination of `Key`/`Value`.
– *The following properties should hold at call time:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)
– *The following properties should hold upon exit:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_consult_b(Key,Value)`
– *Description:* Search for combination of `Key`/`Value` matching pattern. Pattern is supplied as `Value`.
– *The following properties should hold at call time:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)
– *The following properties should hold upon exit:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)

dht_consult_nb/2: PREDICATE

Looks up local database for combination of `Key`/`Value` of following type `Key`/1 that takes `Value` as argument. The combination of `Key`/`Value` is only looked up - no modifications are done in any case (whether a match was found or not). As in case with `dht_extract_nb/2`, this predicate does not block while searching for matching values.

Usage 1: `dht_consult_nb(Key,Value)`
– *Description:* Search for exact combination of `Key`/`Value`.
– *The following properties should hold at call time:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)
– *The following properties should hold upon exit:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)

Usage 2: `dht_consult_nb(Key,Value)`
– *Description:* Search for combination of `Key`/`Value` matching pattern. Pattern is supplied as `Value`.
– *The following properties should hold at call time:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)
– *The following properties should hold upon exit:*
`Key` is an atom. (basic_props:atm/1)
`Value` is ground. (basic_props:gnd/1)

dht_key_hash/2: PREDICATE

A general read-interface to auxiliary database, in order to perform reverse search: 'get set of keys by corresponding value of hash function'.

Usage 1: `dht_key_hash(Key,KeyHash)`

– *Description:* The most general case (when both arguments are free) is searching for any key-hash pair stored in auxiliary database.
– *The following properties should hold at call time:*

`Key` is a free variable. (term_typing:var/1)

`KeyHash` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Key` is ground. (basic_props:gnd/1)

`KeyHash` is an integer. (basic_props:int/1)

Usage 2: `dht_key_hash(Key,KeyHash)`

– *Description:* Here `Key` is ground, which in turn leads to search for information on concrete predicate.
– *The following properties should hold at call time:*

`Key` is ground. (basic_props:gnd/1)

`KeyHash` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`Key` is ground. (basic_props:gnd/1)

`KeyHash` is an integer. (basic_props:int/1)

Usage 3: `dht_key_hash(Key,KeyHash)`

– *Description:* On the contrary to previous example, this one has only second argument `KeyHash` ground. Therefore, this type of call would search for keys, that where mapped into given value of hash-function.
– *The following properties should hold at call time:*

`Key` is a free variable. (term_typing:var/1)

`KeyHash` is an integer. (basic_props:int/1)

– *The following properties should hold upon exit:*

`Key` is ground. (basic_props:gnd/1)

`KeyHash` is an integer. (basic_props:int/1)

Usage 4: `dht_key_hash(Key,KeyHash)`

– *Description:* Finally, calling `dht_key_hash` with both arguments ground is similar to straight-forward check on auxiliary database, or to asking a question: "Does auxiliary database has any information on this key, which is mapped into that hash value?"
– *The following properties should hold at call time:*

`Key` is ground. (basic_props:gnd/1)

`KeyHash` is an integer. (basic_props:int/1)

200 Configuration module

Author(s): Arsen Kostenko.

This is the initial system-wide configuration storage module. All the configurations stored here are represented by command-line arguments or corresponding terms, depending on the server launching style.

200.1 Usage and interface (dht_config)

- **Library usage:**
  `:- use_module(library(dht_config)).`
- **Exports:**
  - *Predicates:*
    `hash_power/1`, `dht_set_hash_power/1`, `dht_s2c_port/1`, `dht_set_s2c_port/1`, `dht_s2c_threads/1`, `dht_set_s2c_threads/1`, `dht_s2s_port/1`, `dht_set_s2s_port/1`, `dht_s2s_threads/1`, `dht_set_s2s_threads/1`, `dht_join_host/1`, `dht_set_join_host/1`, `dht_server_id/1`, `dht_set_server_id/1`, `dht_server_host/1`, `dht_set_server_host/1`.

200.2 Documentation on exports (dht_config)

hash_power/1: PREDICATE

Get/check power of currently running DHT. Hash function has `(2**m)` values, therefore `hash_power/1` returns/checks `m`.

Usage 1: `hash_power(Power)`

- *Description:* Bound `Power` to value of currently running DHT power.
- *The following properties should hold at call time:*
  `Power` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  `Power` is an integer. (basic_props:int/1)

Usage 2: `hash_power(Power)`

- *Description:* Check whether `Power` is equal to power of the currently running DHT.
- *The following properties should hold at call time:*
  `Power` is an integer. (basic_props:int/1)

dht_set_hash_power/1: PREDICATE

Set initial power of hash function.

Usage: `dht_set_hash_power(Power)`

- *Description:* Set `m` for `(2**m)` formula.
- *The following properties should hold at call time:*
  `Power` is an integer. (basic_props:int/1)

dht_s2c_port/1: PREDICATE

Get/check server to client communication port.

Usage 1: `dht_s2c_port(Port)`

- *Description:* Set value of `Port` to currently used port number for server-to-client communication.
- *The following properties should hold at call time:*

  `Port` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*

  `Port` is an integer. (basic_props:int/1)

Usage 2: `dht_s2c_port(Port)`

- *Description:* Check whether value of `Port` is equal to currently used port number for server-to-client communication.
- *The following properties should hold at call time:*

  `Port` is an integer. (basic_props:int/1)

dht_set_s2c_port/1: PREDICATE

Set port for server-to-client communication.

dht_s2c_threads/1: PREDICATE

Get/check number of threads for server-to-client communication.

dht_set_s2c_threads/1: PREDICATE

Set number of threads for server-to-client communication.

dht_s2s_port/1: PREDICATE

Get/check server-to-server communication port.

dht_set_s2s_port/1: PREDICATE

Set server-to-server communication port.

dht_s2s_threads/1: PREDICATE

Get/check number of threads for server-to-server communication.

dht_set_s2s_threads/1: PREDICATE

Set number of threads for server-to-server communication.

dht_join_host/1: PREDICATE

Get/check address of the host to join.

dht_set_join_host/1: PREDICATE
Set address of the host to join.

dht_server_id/1: PREDICATE
Get/check node number of current server.

dht_set_server_id/1: PREDICATE
Set node number of current server.

dht_server_host/1: PREDICATE
Get/check address of the current server

dht_set_server_host/1: PREDICATE
Set address of the current server

201 Tiny module with miscellaneous functions

Author(s): Arsen Kostenko.

This module holds just two predicates at the moment: `write_pr/2` and `read_pr`. In both of them the '_pr' suffix is standing for 'predicate', which in turn means that both of them are intended for transportation of predicates from one environment to another.

201.1 Usage and interface (`dht_misc`)

- **Library usage:**
  `:- use_module(library(dht_misc)).`
- **Exports:**
  - *Predicates:*
    `write_pr/2`, `read_pr/2`.
- **Other modules used:**
  - *System library modules:*
    `fastrw`.

201.2 Documentation on exports (`dht_misc`)

write_pr/2: PREDICATE

`write_pr/2` is a straight-forward wrapping around the `fast_write/2` predicate, without any checks on arguments The sole purpose is to allow usage of various ways of writing to streams without changing entire code of DHT.

Usage: `write_pr(Stream,Term)`

- *Description:* Write the value of the `Term` into stream provided by `Stream`.
- *The following properties should hold at call time:*

  `Stream` is ground. (basic_props:gnd/1)

  `Term` is any term. (basic_props:term/1)

read_pr/2: PREDICATE

Similarly to the previous predicate, this one serves currently as a wrap around the `fast_read/2` predicate, and was implemented with the same purpose - to allow transparent switching to different stream reading systems.

Usage 1: `read_pr(Stream,Term)`

- *Description:* Read stream represented by `Stream` looking for presence of pattern given by `Term`. If none found, the predicate does not block. Result found may be non-fully ground.
- *The following properties should hold at call time:*

  `Stream` is ground. (basic_props:gnd/1)

  `Term` is any term. (basic_props:term/1)

Usage 2: `read_pr(Stream,Term)`

– *Description:* Same as previous, with only modification - result found may be ground as well.
– *The following properties should hold at call time:*

`Stream` is ground. (basic_props:gnd/1)

`Term` is any term. (basic_props:term/1)

– *The following properties should hold upon exit:*

`Stream` is ground. (basic_props:gnd/1)

`Term` is ground. (basic_props:gnd/1)

Usage 3: `read_pr(Stream,Term)`

– *Description:* Scan stream `Stream` for presence of concrete (exact) predicate given by value of `Term`. `Term` is fully bound.
– *The following properties should hold at call time:*

`Stream` is ground. (basic_props:gnd/1)

`Term` is ground. (basic_props:gnd/1)

– *The following properties should hold upon exit:*

`Stream` is ground. (basic_props:gnd/1)

`Term` is ground. (basic_props:gnd/1)

202 Constraint programming over finite domains

Author(s): José Manuel Gómez Pérez, Manuel Carro.

This package is a very preliminary implementation of a finite domain solver. Examples can be found in the source and library directories.

- SEND + MORE = MONEY:

```
:- use_package(fd).
:- use_module(library(prolog_sys), [statistics/2]).
:- use_module(library(format)).


smm(SMM) :-
        statistics(runtime,_),
        do_smm(SMM),
        statistics(runtime,[_, Time]),
        format("Used ~d milliseconds~n", Time).

do_smm(X) :-
        X = [S,E,N,D,M,O,R,Y],
        X in 0 .. 9,
        all_different(X),
        M .>. 0,
        S .>. 0,
        1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E
        .=. 10000*M + 1000*O + 100*N + 10*E + Y,
        labeling(X).
```

- Queens:

```
:- use_package(fd).
:- use_module(library(prolog_sys), [statistics/2]).
:- use_module(library(format)).
:- use_module(library(aggregates)).
:- use_module(library(lists), [length/2]).


queens(N, Qs) :-
        statistics(runtime,_),
        do_queens(N, Qs),
        statistics(runtime,[_, Time]),
        format("Used ~d milliseconds~n", Time).

do_queens(N, Qs):-
        constrain_values(N, N, Qs),
        all_different(Qs),!,
        labeling(Qs).

constrain_values(0, _N, []).
constrain_values(N, Range, [X|Xs]):-
        N > 0,
        X in 1 .. Range,
```

```
        N1 is N - 1,
        constrain_values(N1, Range, Xs),
        no_attack(Xs, X, 1).

no_attack([], _Queen, _Nb).
no_attack([Y|Ys], Queen, Nb):-
        Nb1 is Nb + 1,
        no_attack(Ys, Queen, Nb1),
        Queen .<>. Y + Nb,
        Queen .<>. Y - Nb.
```

202.1 Usage and interface (fd_doc)

- **Library usage:**
  `:- use_package(fd).`
  or
  `:- module(...,...,[fd]).`
- **Exports:**
  - *Predicates:*
    `labeling/1`, `pitm/2`, `choose_var/3`, `choose_free_var/2`, `choose_var_nd/2`, `choose_value/2`, `retrieve_range/2`, `retrieve_store/2`, `glb/2`, `lub/2`, `bounds/3`, `retrieve_list_of_values/2`.
  - *Regular Types:*
    `fd_item/1`, `fd_range/1`, `fd_subrange/1`, `fd_store/1`, `fd_store_entity/1`.
- **New operators defined:**
  `.=./2` [700,xfx], `.<>./2` [700,xfx], `.<./2` [700,xfx], `.=<./2` [700,xfx], `.>./2` [700,xfx], `.>=./2` [700,xfx], `../2` [500,yfx], `.&./2` [600,xfy], `in/2` [700,xfy].

202.2 Documentation on exports (fd_doc)

fd_item/1: REGTYPE

Usage: `fd_item(FD_item)`

- *Description:* `FD_item` is a finite domain entity, i.e. either a finite domains variable or an integer.

fd_range/1: REGTYPE

Usage: `fd_range(FD_range)`

- *Description:* `FD_range` is the range of a finite domain entity.

fd_subrange/1: REGTYPE

Usage:

- *Description:* A subrange is a pair representing a single interval.

fd_store/1: REGTYPE

Usage: `fd_store(FD_store)`

– *Description:* `FD_store` is a representation of the constraint store of a finite domain entity.

fd_store_entity/1: REGTYPE

Usage:

– *Description:* Representation of primitive constraints.

labeling/1: PREDICATE

Usage: `labeling(Vars)`

– *Description:* Implements the labeling process. Assigns values to the input variables `Vars`. On exit all variables are instantiated to a consistent value. On backtracking, the predicate returns all possible assignments. No labeling heuristics implemented so far, i.e. variables are instantiated in their order of appearance.

– *The following properties should hold at call time:*

`Vars` is a list of `fd_item`s. (basic_props:list/2)

pitm/2: PREDICATE

Usage: `pitm(V,MiddlePoint)`

– *Description:* Returns in `MiddlePoint` the intermediate value of the range of `V`. In case `V` is a ground integer value the returned value is `V` itself.

– *The following properties should hold at call time:*

`V` is currently a term which is not a free variable. (term_typing:nonvar/1)

`MiddlePoint` is a free variable. (term_typing:var/1)

`V` is a finite domain entity, i.e. either a finite domains variable or an integer. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_item/1)

`MiddlePoint` is an integer. (basic_props:int/1)

choose_var/3: PREDICATE

Usage: `choose_var(ListOfVars,Var,RestOfVars)`

– *Description:* Returns a finite domain item `Var` from a list of fd items `ListOfVars` and the rest of the list `RestOfVars`in a deterministic way. Currently it always returns the first item of the list.

– *The following properties should hold at call time:*

`ListOfVars` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Var` is a free variable. (term_typing:var/1)

`RestOfVars` is a free variable. (term_typing:var/1)

`ListOfVars` is a list of `fd_item`s. (basic_props:list/2)

`Var` is a finite domain entity, i.e. either a finite domains variable or an integer. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_item/1)

`RestOfVars` is a list of `fd_item`s. (basic_props:list/2)

choose_free_var/2: PREDICATE

Usage: `choose_free_var(ListOfVars,Var)`

– *Description:* Returns a free variable `Var` from a list of fd items `ListOfVars`. Currently it always returns the first free variable of the list.

– *The following properties should hold at call time:*

`ListOfVars` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Var` is a free variable. (term_typing:var/1)

`ListOfVars` is a list of `fd_item`s. (basic_props:list/2)

`Var` is a free variable. (term_typing:var/1)

choose_var_nd/2: PREDICATE

Usage: `choose_var_nd(ListOfVars,Var)`

– *Description:* Returns non deterministically an fd item `Var` from a list of fd items `ListOfVars` .

– *The following properties should hold at call time:*

`ListOfVars` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Var` is a free variable. (term_typing:var/1)

`ListOfVars` is a list of `fd_item`s. (basic_props:list/2)

`Var` is a finite domain entity, i.e. either a finite domains variable or an integer. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_item/1)

choose_value/2: PREDICATE

Usage: `choose_value(Var,Value)`

– *Description:* Produces an integer value `Value` from the domain of `Var`. On backtracking returns all possible values for `Var`.

– *The following properties should hold at call time:*

`Var` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Value` is a free variable. (term_typing:var/1)

`Var` is a finite domain entity, i.e. either a finite domains variable or an integer. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_item/1)

`Value` is an integer. (basic_props:int/1)

retrieve_range/2: PREDICATE

Usage: `retrieve_range(Var,Range)`

– *Description:* Returns in `Range` the range of an fd item `Var`.

– *The following properties should hold at call time:*

`Var` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Range` is a free variable. (term_typing:var/1)

`Var` is a free variable. (term_typing:var/1)

`Range` is the range of a finite domain entity. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_range/1)

retrieve_store/2: PREDICATE

Usage: `retrieve_store(Var,Store)`

– *Description:* Returns in `Store` a representation of the constraint store of an fd item `Var`.

– *The following properties should hold at call time:*

`Var` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Store` is a free variable. (term_typing:var/1)

`Var` is a free variable. (term_typing:var/1)

`Store` is a representation of the constraint store of a finite domain entity. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_store/1)

glb/2: PREDICATE

Usage: `glb(Var,LowerBound)`

– *Description:* Returns in `LowerBound` the lower bound of the range of `Var`.

– *The following properties should hold at call time:*

`Var` is currently a term which is not a free variable. (term_typing:nonvar/1)

`LowerBound` is a free variable. (term_typing:var/1)

`Var` is a finite domain entity, i.e. either a finite domains variable or an integer. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_item/1)

`LowerBound` is an integer. (basic_props:int/1)

lub/2: PREDICATE

Usage: `lub(Var,UpperBound)`

– *Description:* Returns in `UpperBound` the upper bound of the range of `Var`.

– *The following properties should hold at call time:*

`Var` is currently a term which is not a free variable. (term_typing:nonvar/1)

`UpperBound` is a free variable. (term_typing:var/1)

`Var` is a finite domain entity, i.e. either a finite domains variable or an integer. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_item/1)

`UpperBound` is an integer. (basic_props:int/1)

bounds/3: PREDICATE

Usage: `bounds(Var,LowerBound,UpperBound)`

– *Description:* Returns in `LowerBound` and `UpperBound` the lower and upper bounds of the range of `Var`.

– *The following properties should hold at call time:*

`Var` is currently a term which is not a free variable. (term_typing:nonvar/1)

`LowerBound` is a free variable. (term_typing:var/1)

`UpperBound` is a free variable. (term_typing:var/1)

`Var` is a finite domain entity, i.e. either a finite domains variable or an integer. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_item/1)

`LowerBound` is an integer. (basic_props:int/1)

`UpperBound` is an integer. (basic_props:int/1)

retrieve_list_of_values/2: PREDICATE

Usage: `retrieve_list_of_values(Var,ListOfValues)`

– *Description:* Returns in `ListOfValues` an enumeration of al the values in the range of `Var`

– *The following properties should hold at call time:*

`Var` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ListOfValues` is a free variable. (term_typing:var/1)

`Var` is a finite domain entity, i.e. either a finite domains variable or an integer. (user(/home/ciaotester/tests/auto/CiaoDE/ciao/contrib/fd/fd_doc):fd_item/1)

`ListOfValues` is a list of `int`s. (basic_props:list/2)

203 Dot generator

Author(s): Claudio Ochoa.

This module generates a dot file representing a graph. Nodes and edges can contain labels

203.1 Usage and interface (gendot)

- **Library usage:**
  `:- use_module(library(gendot)).`
- **Exports:**
  - *Predicates:*
    `gendot/3`.
- **Other modules used:**
  - *System library modules:*
    `terms`, `format`.

203.2 Documentation on exports (gendot)

gendot/3: PREDICATE

Usage: `gendot(L,Filename,Type)`

- *Description:* Generates a dot file from a list `L` representing a graph. It receives the basename `Filename` (without extension) of the output .dot file. Each element of `L` is a node in the graph, represented by a tuple (Identifier, Label, Edges), where Edges is a list of the outgoing edges of the current node, and each element of Edges is either an Identifier ot a tuple (Identifier,Label). In all cases, Labels are atoms. `Type` indicates the type of graph. If the graph is a Tree, then final nodes are represented by boxes. In all other cases, nodes are represented by circles
- *The following properties should hold at call time:*

  `L` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Filename` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Type` is currently a term which is not a free variable. (term_typing:nonvar/1)

204 Printing graphs using gnuplot as auxiliary tool

Author(s): David Trallero Mena.

This library uses `gnuplot` for printing graphs.

User-friendly predicates to generate data plots are provided, as well as predicates to set the general options which govern the generation of such plots. If no options is specified, global ones are used for data plots generation.

Several files can be generated as temporary files. A BaseName is required for generating the temporaries files. Data files name will be created from BaseName `+` number `+` .dat. The BaseName `+ ".plot"` will be the name used for `gnuplot` tool.

A list of pairs of list of pairs of the from (X,Y) and Local Option value is provided to the main predicate as data. In other words DataList = [(CurveDataList,LocalOptions), (CurveDataList1,LocalOptions1) ...]. Additionaly (function(String) , LocalOptions) can be used for adding a curve to the plot (imagine you want to compare your result with 'x=y').

LocalOptions of the DataList are options that are applied to the curve, as for example, if we print the curve with lines, or the title in the legend, etc. GlobalOptions are referred to the plot options, like title in x or y axis, etc.

204.1 Usage and interface (`gnuplot`)

- **Library usage:**
  `:- use_module(library(gnuplot)).`
- **Exports:**
  - *Predicates:*
    `get_general_options/1`, `set_general_options/1`, `generate_plot/2`, `generate_plot/3`.
- **Other modules used:**
  - *System library modules:*
    `lists`, `write`, `system`.

204.2 Documentation on exports (`gnuplot`)

get_general_options/1: PREDICATE

Usage: `get_general_options(X)`

- *Description:* Get the general options of the graphic that will be plotted
- *The following properties should hold at call time:*
  `X` is a free variable. (term_typing:var/1)
- *The following properties should hold upon exit:*
  `X` is a list. (basic_props:list/1)

set_general_options/1: PREDICATE

Usage: `set_general_options(X)`

– *Description:* Get the general options of the graphic that will be plotted. Possible options are:
  - format(A) Specify the format of points
  - nokey Legend is no represented
  - nogrid No grid
  - grid An smooth grid is shown
  - label(L , (X,Y)) Put Label L at point (X,Y)
  - xlabel(A) Label of X-Axis
  - ylabel(A) Label of Y-Axis
  - xrange(A,B) Define the X range representation
  - yrange(A,B) Define the Y range representation
  - title(A) Title of the plot
  - key(A) define the key (for example [left,box], left is the position, box indicates that a box should be around)
  - term_post(A) define the postscript terminal. A is a list of atoms.
  - size(A,B) specify the size of the plot (A,B float numbers)
  - autoscale autoscale the size of the plot
  - autoscale(A) autoscale the argument (for example: autoscale(x))
– *The following properties should hold at call time:*
  `X` is a list. (basic_props:list/1)

generate_plot/2: PREDICATE

Usage: `generate_plot(BaseName,DataList)`

– *Description:* This predicates generate a '`BaseName` + .ps' postscript file using each element of `DataList` as pair of list of pairs and local options, i.e., (list((X,Y)), LocalOptions), in which X is the position in X-Axis and Y is the position in Y-Axis. Nevertheless, each element of `DataList` can be a list of pairs instead of a pair for commodity. `gnuplot` is used as auxiliary tool. Temporary files '`BaseName` + N.dat' are generated for for every list of pairs, and '`BaseName` + .plot' is de file used by gnuplot. The local options can be:
  - with(Option) Tells how the curve will be represented. Option can b line, dots, boxes, impulses, linespoints. This option HAVE TO BE the last one
  - title(T) Put the name of the curve in the legend to T
– *The following properties should hold at call time:*
  `BaseName` is currently instantiated to an atom. (term_typing:atom/1)
  `DataList` is a list of `pair`s. (basic_props:list/2)
– *The following properties should hold upon exit:*
  `BaseName` is currently instantiated to an atom. (term_typing:atom/1)
  `DataList` is a list of `pair`s. (basic_props:list/2)

generate_plot/3: PREDICATE

Usage: `generate_plot(BaseName,DataList,GeneralOptions)`

– *Description:* It is the same as generate_plot/2 but `GeneralOptions` are used as the general options of the plot. Look at predicate set_general_options for detailed description of possible options

– *The following properties should hold at call time:*
  `BaseName` is currently instantiated to an atom. (term_typing:atom/1)
  `DataList` is a list of `pair`s. (basic_props:list/2)
  `GeneralOptions` is a list. (basic_props:list/1)
– *The following properties should hold upon exit:*
  `BaseName` is currently instantiated to an atom. (term_typing:atom/1)
  `DataList` is a list of `pair`s. (basic_props:list/2)
  `GeneralOptions` is a list. (basic_props:list/1)

205 Lazy evaluation

Author(s): Amadeo Casas (`http://www.cs.unm.edu/~amadeo`, University of New Mexico).

This library package allows the use of lazy evaluation in a Ciao module/program.

Lazy Evaluation is a program evaluation technique used particularly in functional languages. When using lazy evaluation, an expression is not evaluated as soon as it is assigned, but rather when the evaluator is forced to produce the value of the expression. Although the `when` or `freeze` control primitives present in many modern logic programming systems are more powerful than lazy evaluation, they lack the simplicity of use and cleaner semantics of functional lazy evaluation.

The objective of this package is to allow evaluating the functions lazily. Functions are the subset of relations (predicates) which have a designated argument through which a single output is obtained for any set of inputs (the other arguments). In logic programming systems which have syntactic support for functions (including Ciao), functions are typically translated to predicates whose `last` argument is designated as a (single value) output and the rest as inputs.

In our proposal, a function can be declared as lazy via the following declaration:

```
:- lazy fun_eval f/N.
```

This function could be represented as:

```
:- lazy fun_eval f(~_,_,_,_).
```

where `~` indicates the argument through which the single output will be obtained. Another possible representation may be:

```
:- lazy fun_return f(~_,_,_,_).
```

In order to achieve the intended behavior, the execution of each function declared as lazy is suspended until the return value of the function is needed.

A simple example of the use of lazy evaluation would be the definition of a function which returns the (potentially) infinite list of integers starting with a given one:

```
:- lazy fun_eval nums_from/1.
nums_from(X) := [X | nums_from(X+1)].
```

While lazy functions certainly increase the overhead in the execution, they also allow the user to develop in an easy way predicates which can handle infinite terms, and this is the main advantage of the proposed functionality.

Lazy evaluation can be also a better option than eager evaluation when a function in a different module is used and it returns a big amount of data. As an example, we have the following module `module1`:

```
:- module(module1, [test/1], [fsyntax, lazy, hiord]).

:- use_module(library(lazy(lazy_lib)), [nums_from/2, takeWhile/3]).
:- use_module(module2, [squares/2]).
:- use_module(library(arithpreds)).

:- fun_eval test/0.
test := ~takeWhile((''(X) :- X < 10000), ~squares(~nums_from(1))).
```

and another module `module2`:

```
:- module(module1, [test/1], [fsyntax, lazy, hiord]).

:- use_module(library(lazy(lazy_lib)), [nums_from/2, takeWhile/3]).
:- use_module(module2, [squares/2]).
```

```
:- use_module(library(arithpreds)).

:- fun_eval test/0.
test := ~takeWhile((''(X) :- X < 10000), ~squares(~nums_from(1))).
```

Function `test/0` in module `m1` needs to execute function `squares/1`, in module `m2`, which will return a very long list (in the case of this example this list will be infinite, but the conclusions also apply with finite but long lists). If `squares/1` were executed eagerly then the entire list would be returned, to immediately execute the `take/2` function with the entire list, but creating this intermediate result is wasteful in terms of memory requirements. In order to solve this problem, the `squares/1` function could be moved to module `m1` and merged with `take/2` (or, also, they could exchange a size parameter). But rearranging the program is not always possible and may perhaps complicate other aspects of the overall program design.

If instead the `squares/1` function is evaluated lazily, it is possible to keep the definitions unchanged and in different modules and there will be a smaller memory penalty for storing the intermediate result. As more values are needed by the `take/2` function, more values in the list returned by `squares/1` are built (in this example, only 10 values). These values that have been consumed and passed over will be recovered by the garbage collector and the corresponding memory freed. The query:

```
?- test(X).
```

will compute `X` = [1,4,9,16,25,36,49,64,81,100].

A library of useful functions has been added to this package to allow the programmer to develop lazy functions easily and with a well-defined syntax. This library is called **lazy_lib.pl** and it provides the following functions:

- nums_from(+X,-List): `List` is unified with an infinite list of successive numbers starting in `X`.
- nums_from_inc(+X,+Y,-List): `List` is unified with an infinite list of successive numbers starting in `X` with an increment of `Y`.
- repeat(+X,-List): `List` is unified with an infinite list of the term `Y`.
- cycle(+X,-List): `List` is unified with an infinite list of the term `Y` repeated infinite times.
- take(+X,+ListA,-ListR): `ListR` is unified with the first `X` elements of the infinite list `ListA`.
- takeWhile(+P,+ListA,-ListR): `ListR` is unified with the first elements of the infinite list `ListA` while the condition `P` is true.
- drop(+X,+ListA,-ListR): `ListR` is unified with the infinite list `ListA` dropping the first `X` elements.
- dropWhile(+P,+ListA,-ListR): `ListR` is unified with the infinite list `ListA` dropping the first elements while the condition `P` is true.
- splitAt(+X,+ListA,-Res): `Res` is unified with a tuple of lists where the first list is composed by the first `X` elements of the list `ListA` and the second list is composed by the rest of the elements of `ListA`.
- span(+P,+ListA,-Res): `Res` is unified with a tuple of lists where the first list is composed by the elements of `ListA` which verify the condition `P` and the second list is composed by the rest of the elements of the initial list.
- tail(+ListA,-ListR): `ListR` is unified with the tail of the infinite list `ListA`.
- lazy_map(+ListA,+P,-ListR): Version of the map/3 predicate to be executed lazily.
- lazy_foldl(+ListA,+X,+P,-ListR): Version of the foldl/3 predicate to be executed lazily.
- zipWith(+P,+ListA,+ListB,-ListR): `ListR` is a list whose elements are calculated from the function `P` and the elements of input lists `ListA` and `ListB` occuring at the same position in both lists.

205.1 Usage and interface (lazy_doc)

- **Library usage:**
  `:- use_package(lazy).`
  or
  `:- module(...,...,[lazy]).`
- **New operators defined:**
  `lazy/1` [1170,fx].
- **Other modules used:**
  - *System library modules:*
    `freeze/freeze`.

205.2 Other information (lazy_doc)

The translation of the code in order to execute it lazily is explained below.

A sentence translation is provided to handle the `lazy` directives. The translation of a lazy function into a predicate is done in two steps. First, the function is converted into a predicate (using the fsyntax package). Then, the resulting predicate is transformed to suspend its execution until the value of the last variable (i.e., the output variable) is needed. This suspension is achieved by the use of the `freeze/1` control primitive that many modern logic programming systems implement quite efficiently (`block` or `when` declarations can obviously also be used, but we explain the transformation in terms of `freeze` because it is more widespread). The translation will rename the original predicate to an internal name and add a bridge predicate with the original name which invokes the internal predicate through a call to `freeze/1`. This will delay the execution of the internal predicate until its result is required, which will be detected as a binding (i.e., demand) of its output variable.

We show now an example of the use of lazy evaluation, and how a lazy function is translated by this package. The following code returns an (infinite) list of fibonacci numbers:

```
:- lazy fun_eval fiblist/0.
fiblist := [0, 1 | ~zipWith(add, FibL, ~tail(FibL))]
        :- FibL = fiblist.
```

which is translated into:

```
fiblist(X) :-
          freeze(X, 'fiblist_$$lazy$$'(X)).

'fiblist_$$lazy$$'([0, 1 | Rest]) :-
          fiblist(FibL),
          tail(FibL, T),
          zipWith(add, FibL, T, Rest).
```

In the `fiblist` function defined, any element in the resulting infinite list of fibonacci numbers can be referenced, as for example, `nth(X, ~fiblist, Value).`. The other functions used in the definition are `tail/2` and `zipWith/3`. These two functions can be found in the **lazy_lib.pl** runtime file.

206 Programming MYCIN rules

Author(s): Angel Fernandez Pineda.

MYCIN databases are declared as Prolog modules containing mycin rules. Those rules are given a *certainty factor* (*CF*) which denotates an expert's credibility on that rule:

- A value of -1 stands for *surely not*.
- A value of 1 stands for *certainly*.
- A value of 0 stands for *I don't know*.

Intermediate values are allowed.

Mycin rules work on a different way as Prolog clauses: a rule will never fail (in the Prolog sense), it will return a certainty value instead. As a consequence **all** mycin rules will be explored during inference, so the order in which rules are written is not significant. For this reason, the usage of the Prolog *cut* (!) is discouraged.

206.1 Usage and interface (`mycin_doc`)

- **Library usage:**
  In order to declare a mycin database you must include the following declaration as the first one in your file:

```
:- mycin(MycinDataBaseName).
```

- **New declarations defined:**
  `export/1`.

206.2 Documentation on new declarations (`mycin_doc`)

export/1: DECLARATION

This directive allows a given mycin predicate to be called from Prolog programs. The way in which mycin rules are called departs from Prolog ones. For instance, the followin mycin predicate:

```
:- export p/1.
```

must be called from Prolog Programs as: `mycin(p(X),CF)`, where CF will be binded to the resulting certainty factor. Obviously, the variables on *P/1* may be instantiated as you wish. Since the Prolog predicate *mycin/2* may be imported from several mycin databases, it is recommended to fully qualify those predicate goals. For example : `mydatabase:mycin(p(X),CF)`.

Usage: :- `export(Spec)`.

– *Description:* `Spec` will be a callable mycin predicate.

206.3 Known bugs and planned improvements (`mycin_doc`)

- Not fully implemented.
- Dynamic mycin predicates not implemented: open question.
- Importation of user-defined mycin predicates requires further design. This includes importation of mycin databases from another mycin database.

207 The Ciao Profiler

Author(s): Edison Mera.

The Ciao profiler provides a high-level, flexible way to mark a predicate (or a literal) for profiling. This is done by using declarations to indicate if a program element must be instrumented or not.

By default, if the user does not specify anything, no predicate inside the module will be instrumented as cost center for profiling. The use of at least one declaration saying that a specific predicate must be instrumented overrides this behavior.

The declaration is as follows:

```
:- cost_center pred1/Arity1, ... predN/ArityN.
```

where `pred1/Arity1`, ..., `predN/ArityN` are the predicates to be instrumented as cost centers. They can be separated by commas or the can be in a list.

By default the engine hooks of all defined cost center are active. The declaration:

```
(predN/ArityN,nohooks)
```

will deactivate them.

Another useful declaration makes possible to indicate that a given predicate is not going to be instrumented as cost center:

```
:- no_cost_center pred1/Arity1, ... predN/ArityN.
```

where `pred1/Arity1`, ..., `predN/ArityN` are the predicates that will not be instrumented as cost centers. There are two options (as in the previous case): write one assertion for each predicate or declare more than one predicate (separated by commas or in a list) in only one assertion.

The following assertions define the behavior of all the predicates of the module:

```
:- all_cost_center.
```

```
:- all_no_cost_center.
```

They specify respectively that all the predicates in the module will be instrumented as cost centers and that no predicate in the module will be instrumented as cost center. In the first assertion the engine hooks of all defined cost centers are active. The declaration `:- all_cost_center(nohooks).` will deactivate them.

Cost centers can be also defined at literal level replacing the literal by the declaration:

```
:- cost_center(name_cc, literal)
```

where `literal` is the program literal and `name_cc` is the name of its associated cost center. At predicate level the name of both cost center and predicate are equal.

207.1 Usage and interface (`profiler_doc`)

- **Library usage:**
  The Ciao profiler is used by including `profiler` in the package inclusion list of a module, or by means of an explicit `:- use_package(profiler).` directive.

208 ProVRML - a Prolog interface for VRML

Author(s): Göran Smedbäck, Manuel Carro (some changes), The CLIP Group.

`ProVRML` is Prolog library to handle VRML code. The library consists of modules to handle the tokenising, that is breaking the VRML code into smaller parts that can be analysed further. The further analysis will be the parsing. This is a complex part of the library and consists of several modules to handle errors and value check. When the parsing is done we have the Prolog terms of the VRML code. The terms are quite similar to the origin VRML code and can easily be read if you recognise that syntax.

This Prolog terms of the VRML code is then possible to use for analysis, reconstruction, reverse enginering, building blocks for automatic generation of VRML code. There are several possibilities and these are only some of them.

When you are done with the Prolog terms for the code, you would probably want to reverse the action and return to VRML code. This is done with the code generation modules. These are built up in more or less the same manner as the parser modules.

208.1 Usage and interface (`provrml`)

- **Library usage:**
  `:- use_module(library(provrml)).`
- **Exports:**
  - *Predicates:*
    `vrml_web_to_terms/2`, `vrml_file_to_terms/2`, `vrml_web_to_terms_file/2`, `vrml_file_to_terms_file/2`, `terms_file_to_vrml/2`, `terms_file_to_vrml_file/2`, `terms_to_vrml_file/2`, `terms_to_vrml/2`, `vrml_to_terms/2`, `vrml_in_out/2`, `vrml_http_access/2`.
- **Other modules used:**
  - *System library modules:*
    `pillow/http`, `pillow/html`, `provrml/provrml_io`, `provrml/provrml_parser`, `provrml/generator`.

208.2 Documentation on exports (`provrml`)

vrml_web_to_terms/2: PREDICATE

Usage: `vrml_web_to_terms(WEBAddress,Terms)`

- *Description:* Given a address to a VRML-document on the Internet, the predicate will return the prolog-terms.
- *Call and exit should be compatible with:*
  `WEBAddress` is an atom. (basic_props:atm/1)
  `Terms` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*
  `WEBAddress` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Terms` is a free variable. (term_typing:var/1)

vrml_file_to_terms/2: PREDICATE

Usage 1: `vrml_file_to_terms(FileName,Term)`

- *Description:* Given a filename containing a VRML-file the predicate returns the prolog terms corresponding.
- *Call and exit should be compatible with:*

  `FileName` is an atom. (basic_props:atm/1)

  `Term` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Term` is a free variable. (term_typing:var/1)

Usage 2: `vrml_file_to_terms(FileName,Terms)`

- *Description:* Given a filename containing a VRML-file and a filename, the predicate write the prolog terms corresponding to the filename.
- *Call and exit should be compatible with:*

  `FileName` is an atom. (basic_props:atm/1)

  `Terms` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Terms` is currently a term which is not a free variable. (term_typing:nonvar/1)

vrml_web_to_terms_file/2: PREDICATE

Usage: `vrml_web_to_terms_file(WEBAddress,FileName)`

- *Description:* Given a address to a VRML-document on the Internet and a filename, the predicate will write the prolog_terms to the file.
- *Call and exit should be compatible with:*

  `WEBAddress` is an atom. (basic_props:atm/1)

  `FileName` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `WEBAddress` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

vrml_file_to_terms_file/2: PREDICATE

No further documentation available for this predicate.

terms_file_to_vrml/2: PREDICATE

Usage: `terms_file_to_vrml(FileName,List)`

- *Description:* From a given filename with prologterms on the special format, the predicate returns the corresponding VRML-code.
- *Call and exit should be compatible with:*

  `FileName` is an atom. (basic_props:atm/1)

  `List` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `List` is a free variable. (term_typing:var/1)

terms_file_to_vrml_file/2: PREDICATE

Usage: `terms_file_to_vrml_file(Atom,Atom)`

- *Description:* From a given filename with prologterms on the special format, the predicate writes the corresponding VRML-code to second filename.
- *Call and exit should be compatible with:*

  `Atom` is an atom. (basic_props:atm/1)

  `Atom` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Atom` is currently a term which is not a free variable. (term_typing:nonvar/1)

terms_to_vrml_file/2: PREDICATE

Usage: `terms_to_vrml_file(Term,FileName)`

- *Description:* Given prolog-terms the predicate writes the corresponding VRML-code to the given file.
- *Call and exit should be compatible with:*

  `Term` is an atom. (basic_props:atm/1)

  `FileName` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

terms_to_vrml/2: PREDICATE

Usage: `terms_to_vrml(Term,VRMLCode)`

- *Description:* Given prolog-terms the predicate returns a list with the corresponding VRML-code.
- *Call and exit should be compatible with:*

  `Term` is an atom. (basic_props:atm/1)

  `VRMLCode` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*

  `Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `VRMLCode` is a free variable. (term_typing:var/1)

vrml_to_terms/2: PREDICATE

Usage: `vrml_to_terms(VRMLCode,Terms)`

- *Description:* Given a list with VRML-code the predicate will return the corresponding prolog-terms.
- *Call and exit should be compatible with:*

  `VRMLCode` is a string (a list of character codes). (basic_props:string/1)

  `Terms` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `VRMLCode` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Terms` is a free variable. (term_typing:var/1)

vrml_in_out/2: PREDICATE

Usage: `vrml_in_out(FileName,FileName)`

- *Description:* This is a controll-predicate that given a filename to a VRML-file and a filename, the predicate will read the VRML-code. Transform it to prolog-terms and then transform it back to VRRML-code and write it to the latter file.
- *Call and exit should be compatible with:*

  `FileName` is an atom. (basic_props:atm/1)

  `FileName` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

vrml_http_access/2: PREDICATE

Usage: `vrml_http_access(ReadFilename,BaseFilename)`

- *Description:* Given a web-address to a VRML-file the predicate will load the code, write it first to the second argument with extension '_first.wrl'. Then it transform the code to prolog terms and write it with the extension '.term'. Transform it back to VRML-code and write it to the filename with '.wrl. A good test-predicate.
- *Call and exit should be compatible with:*

  `ReadFilename` is an atom. (basic_props:atm/1)

  `BaseFilename` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `ReadFilename` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `BaseFilename` is currently a term which is not a free variable. (term_typing:nonvar/1)

208.3 Documentation on internals (`provrml`)

read_page/2: PREDICATE

Usage: `read_page(WEBAddress,Data)`

- *Description:* This routine reads a page on the web using pillow routines.
- *Call and exit should be compatible with:*

  `WEBAddress` is an atom. (basic_props:atm/1)

  `Data` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*

  `WEBAddress` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Data` is a free variable. (term_typing:var/1)

209 boundary (library)

Author(s): Göran Smedbäck.

This module offers predicate to check values according to their boundaries and offers lists of possible node ascendents.

209.1 Usage and interface (boundary)

- **Library usage:**
  `:- use_module(library(boundary)).`
- **Exports:**
  - *Predicates:*
    `boundary_check/3`, `boundary_rotation_first/2`, `boundary_rotation_last/2`, `reserved_words/1`, `children_nodes/1`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `provrml/internal_types`, `provrml/provrmlerror`.

209.2 Documentation on exports (boundary)

boundary_check/3: PREDICATE

Usage: `boundary_check(Value_to_check,Init_value,Bound)`

- *Description:* This predicate check the boundaries of the given value according to the boudary values. If the value is wrong according to the boundaries, the value is checked according to the initial value given. If the value continues to be wrong, an error will be raised accordingly.
- *Call and exit should be compatible with:*
  `Value_to_check` is an atom. (basic_props:atm/1)
  `Init_value` is a list of `atm`s. (basic_props:list/2)
  `Bound` is a variable interval. (internal_types:bound/1)
- *The following properties should hold at call time:*
  `Value_to_check` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Init_value` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Bound` is currently a term which is not a free variable. (term_typing:nonvar/1)

boundary_rotation_first/2: PREDICATE

Usage: `boundary_rotation_first(Bound_double,Bound)`

- *Description:* The predicate will extract the first bounds from a double bound.
- *Call and exit should be compatible with:*
  `Bound_double` is a variable interval. (internal_types:bound_double/1)
  `Bound` is a variable interval. (internal_types:bound/1)

– *The following properties should hold at call time:*
`Bound_double` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Bound` is a free variable. (term_typing:var/1)

boundary_rotation_last/2: PREDICATE

Usage: `boundary_rotation_last(Bound_double,Bound)`

– *Description:* The predicate will extract the last bounds from a double bound.
– *Call and exit should be compatible with:*
`Bound_double` is a variable interval. (internal_types:bound_double/1)
`Bound` is a variable interval. (internal_types:bound/1)
– *The following properties should hold at call time:*
`Bound_double` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Bound` is a free variable. (term_typing:var/1)

reserved_words/1: PREDICATE

Usage: `reserved_words(List)`

– *Description:* Returns a list with the reserved words, words prohibited to use in cases not appropiated.
– *Call and exit should be compatible with:*
`List` is a list of `atm`s. (basic_props:list/2)
– *The following properties should hold at call time:*
`List` is a free variable. (term_typing:var/1)

children_nodes/1: PREDICATE

Usage: `children_nodes(List)`

– *Description:* Returns a list of all nodes possible as children nodes.
– *Call and exit should be compatible with:*
`List` is a list of `atm`s. (basic_props:list/2)
– *The following properties should hold at call time:*
`List` is a free variable. (term_typing:var/1)

210 dictionary (library)

Author(s): Göran Smedbäck.

This module contains the fixed dictionary. All the nodes in VRML with their associated fields.

210.1 Usage and interface (dictionary)

- **Library usage:**
  `:- use_module(library(dictionary)).`
- **Exports:**
  - *Predicates:*
    `dictionary/6`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `provrml/internal_types`.

210.2 Documentation on exports (dictionary)

dictionary/6: PREDICATE

Usage 1:
`dictionary(NodeTypeId,AccessType,FieldTypeId,FieldId,Init_value,Boundary)`

- *Description:* To lookup information about the nodes, getting their properties. Note that the type returned for the bound can be of two different types bound or bound_double. The rotation type have one bound for the directions and one for the degree of rotation.
- *Call and exit should be compatible with:*

`NodeTypeId` is an atom.	(basic_props:atm/1)
`AccessType` is an atom.	(basic_props:atm/1)
`FieldTypeId` is an atom.	(basic_props:atm/1)
`FieldId` is an atom.	(basic_props:atm/1)
`Init_value` is a list of `atm`s.	(basic_props:list/2)
`Boundary` is a variable interval.	(internal_types:bound/1)

- *The following properties should hold at call time:*

`Init_value` is a free variable.	(term_typing:var/1)
`Boundary` is a free variable.	(term_typing:var/1)

Usage 2:
`dictionary(NodeTypeId,AccessType,FieldTypeId,FieldId,Init_value,Boundary)`

- *Description:* To lookup information about the nodes, getting their properties. Note that the type returned for the bound can be of two different types bound or bound_double. The rotation type have one bound for the directions and one for the degree of rotation.

– *Call and exit should be compatible with:*
  `NodeTypeId` is an atom. (basic_props:atm/1)
  `AccessType` is an atom. (basic_props:atm/1)
  `FieldTypeId` is an atom. (basic_props:atm/1)
  `FieldId` is an atom. (basic_props:atm/1)
  `Init_value` is a list of `atm`s. (basic_props:list/2)
  `Boundary` is a variable interval. (internal_types:bound_double/1)
– *The following properties should hold at call time:*
  `Init_value` is a free variable. (term_typing:var/1)
  `Boundary` is a free variable. (term_typing:var/1)

211 dictionary_tree (library)

Author(s): Göran Smedbäck.

This module offers a dynamic tree structured dictionary a bit combined with predicates that gives it the useability to be the dictionary for the parser.

211.1 Usage and interface (dictionary_tree)

- **Library usage:**
  `:- use_module(library(dictionary_tree)).`
- **Exports:**
  - *Predicates:*
    `create_dictionaries/1`, `is_dictionaries/1`, `get_definition_dictionary/2`, `get_prototype_dictionary/2`, `dictionary_insert/5`, `dictionary_lookup/5`, `merge_tree/2`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `lists`, `provrml/internal_types`.

211.2 Documentation on exports (dictionary_tree)

create_dictionaries/1: PREDICATE

Usage: `create_dictionaries(Dictionary)`

- *Description:* Returns a dictionary. A general name was used if the user would like to change the code to include more dictionaries.
- *Call and exit should be compatible with:*
  `Dictionary` is a dictionary. (internal_types:dictionary/1)
- *The following properties should hold at call time:*
  `Dictionary` is a free variable. (term_typing:var/1)

is_dictionaries/1: PREDICATE

Usage: `is_dictionaries(Dictionary)`

- *Description:* Is the argument a dictionary is solved by this predicate.
- *Call and exit should be compatible with:*
  `Dictionary` is a dictionary. (internal_types:dictionary/1)

get_definition_dictionary/2: PREDICATE

Usage: `get_definition_dictionary(Dictionary,Tree)`

- *Description:* Returns the definition dictionary (for the moment there is only one dictionary), which is a tree representation.

– *Call and exit should be compatible with:*

`Dictionary` is a dictionary. (internal_types:dictionary/1)

`Tree` is a tree structure. (internal_types:tree/1)

– *The following properties should hold at call time:*

`Dictionary` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Tree` is a free variable. (term_typing:var/1)

get_prototype_dictionary/2: PREDICATE

Usage: `get_prototype_dictionary(Dictionary,Tree)`

– *Description:* Returns the prototype dictionary (for the moment there is only one dictionary), which is a tree representation.

– *Call and exit should be compatible with:*

`Dictionary` is a dictionary. (internal_types:dictionary/1)

`Tree` is a tree structure. (internal_types:tree/1)

– *The following properties should hold at call time:*

`Dictionary` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Tree` is a free variable. (term_typing:var/1)

dictionary_insert/5: PREDICATE

Usage: `dictionary_insert(Key,Type,Field,Dictionary,Info)`

– *Description:* The predicate will search for the place for the Key and return Info, if the element inserted had a post before (same key value) multiple else new. The dictionary is dynamic and do not need output because of using unbinded variables.

– *Call and exit should be compatible with:*

`Key` is an atom. (basic_props:atm/1)

`Type` is an atom. (basic_props:atm/1)

`Field` is any term. (basic_props:term/1)

`Dictionary` is a tree structure. (internal_types:tree/1)

`Info` is an atom. (basic_props:atm/1)

– *The following properties should hold at call time:*

`Key` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Type` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Field` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Dictionary` is currently a term which is not a free variable. (term_typing:nonvar/1)

dictionary_lookup/5: PREDICATE

Usage: `dictionary_lookup(Key,Type,Field,Dictionary,Info)`

– *Description:* The predicate will search for the Key and return Info;defined or undefined accordingly. If defined the fields will be filled as well. The predicate do not insert the element.

– *Call and exit should be compatible with:*

`Key` is an atom.	(basic_props:atm/1)
`Type` is an atom.	(basic_props:atm/1)
`Field` is any term.	(basic_props:term/1)
`Dictionary` is a dictionary.	(internal_types:dictionary/1)
`Info` is an atom.	(basic_props:atm/1)

– *The following properties should hold at call time:*

`Key` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Dictionary` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Info` is a free variable.	(term_typing:var/1)

merge_tree/2: PREDICATE

Usage: `merge_tree(Tree,Tree)`

– *Description:* The predicate can be used for adding a tree dictionary to another one (the second). It will remove equal posts but posts with a slight difference will be inserted. The resulting tree will be the second tree.

– *Call and exit should be compatible with:*

`Tree` is a tree structure.	(internal_types:tree/1)
`Tree` is a tree structure.	(internal_types:tree/1)

– *The following properties should hold at call time:*

`Tree` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Tree` is currently a term which is not a free variable.	(term_typing:nonvar/1)

212 provrmlerror (library)

Author(s): Göran Smedbäck.

This file implements error predicates of different types.

212.1 Usage and interface (provrmlerror)

- **Library usage:**
  `:- use_module(library(provrmlerror)).`
- **Exports:**
  - *Predicates:*
    `error_vrml/1`, `output_error/1`.
- **Other modules used:**
  - *System library modules:*
    `write`.

212.2 Documentation on exports (provrmlerror)

error_vrml/1: PREDICATE

Usage: `error_vrml(Structure)`

- *Description:* Given a structure with the error type as its head with possible arguments, it will write the associated error-text.
- *Call and exit should be compatible with:*
  `Structure` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*
  `Structure` is currently a term which is not a free variable. (term_typing:nonvar/1)

output_error/1: PREDICATE

Usage: `output_error(Message)`

- *Description:* This predicate will print the error message given as the argument. This predicate is used for warnings that only needs to be given as information and not necessarily give an error by the VRML browser.
- *Call and exit should be compatible with:*
  `Message` is a list of `atm`s. (basic_props:list/2)
- *The following properties should hold at call time:*
  `Message` is currently a term which is not a free variable. (term_typing:nonvar/1)

213 field_type (library)

Author(s): Göran Smedbäck.

213.1 Usage and interface (field_type)

- **Library usage:**
  `:- use_module(library(field_type)).`
- **Exports:**
  - *Predicates:*
    `fieldType/1`.

213.2 Documentation on exports (field_type)

fieldType/1: PREDICATE

Usage: `fieldType(FieldTypeId)`

- *Description:* Boolean predicate used to check the fieldTypeId with the defiened.
- *Call and exit should be compatible with:*
  `FieldTypeId` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*
  `FieldTypeId` is currently a term which is not a free variable. (term_typing:nonvar/1)

214 field_value (library)

Author(s): Göran Smedbäck.

214.1 Usage and interface (field_value)

- **Library usage:**
  `:- use_module(library(field_value)).`
- **Exports:**
  - *Predicates:*
    `fieldValue/6`, `mfstringValue/5`.
  - *Properties:*
    `parse/1`.
- **Other modules used:**
  - *System library modules:*
    `lists`, `provrml/provrml_parser`, `provrml/parser_util`, `provrml/provrmlerror`.

214.2 Documentation on exports (field_value)

fieldValue/6: PREDICATE

Usage: `fieldValue(ParseIn,ParseOut,FieldTypeId,FieldValue,L,T)`

- *Description:* The predicate read the fieldValue from the input token stream and return the value of the parsing. The resulting list might be of numbers, strings or VRML code dependnig on the FieldTypeId.
- *Call and exit should be compatible with:*

field_value:parse(ParseIn)	(field_value:parse/1)
field_value:parse(ParseOut)	(field_value:parse/1)
`FieldTypeId` is an atom.	(basic_props:atm/1)
`FieldValue` is a list of `terms`.	(basic_props:list/2)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

- *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)
`FieldTypeId` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldValue` is a free variable.	(term_typing:var/1)

mfstringValue/5: PREDICATE

Usage: `mfstringValue(ParseIn,ParseOut,Value,L,T)`

- *Description:* The predicate is exported for 'EXTERNPROTO' use, where names for locations are given. Reads one string value or multiple stringvalues from a list.

– *Call and exit should be compatible with:*

field_value:parse(ParseIn)	(field_value:parse/1)
field_value:parse(ParseOut)	(field_value:parse/1)
`Value` is a list of `string`s.	(basic_props:list/2)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)
`Value` is a free variable.	(term_typing:var/1)

parse/1: PROPERTY

A property, defined as follows:

```
parse(_1).
parse(parse(In,Out,Env,Dic)) :-
        list(In),
        list(Out),
        environment(Env),
        dictionary(Dic).
```

215 field_value_check (library)

Author(s): Göran Smedbäck.

215.1 Usage and interface (field_value_check)

- **Library usage:**
  `:- use_module(library(field_value_check)).`
- **Exports:**
  - *Predicates:*
    `fieldValue_check/8`, `mfstringValue/7`.
- **Other modules used:**
  - *System library modules:*
    `provrml/provrml_io`, `provrml/generator_util`, `provrml/boundary`, `provrml/tokeniser`, `provrml/generator`, `provrml/parser_util`.

215.2 Documentation on exports (field_value_check)

fieldValue_check/8: PREDICATE

Usage: `fieldValue_check(FieldTypeId,Value,ParseIn,ParseOut,InitValue,Boundary,L,T)`

- *Description:* The predicate read the fieldValue from the input token stream from the ParseIn. Checks of the values will be done in other module but initiated here. The values will be, if correct, collected via the DCG with the out/3 predicate for later output.

  All the predicates have the same meaning as the corresponding predicates in the input module field_value.pl. For more information about the different please see that module.
- *Call and exit should be compatible with:*

`FieldTypeId` is an atom.	(basic_props:atm/1)
`Value` is a list.	(basic_props:list/1)
field_value_check:parse(ParseIn)	(field_value_check:parse/1)
field_value_check:parse(ParseOut)	(field_value_check:parse/1)
`InitValue` is an atom.	(basic_props:atm/1)
`Boundary` is an atom.	(basic_props:atm/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

- *The following properties should hold at call time:*

`FieldTypeId` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Value` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)
`InitValue` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Boundary` is currently a term which is not a free variable.	(term_typing:nonvar/1)

mfstringValue/7: PREDICATE

No further documentation available for this predicate.

216 generator (library)

Author(s): Göran Smedbäck.

216.1 Usage and interface (generator)

- **Library usage:**
  `:- use_module(library(generator)).`
- **Exports:**
  - *Predicates:*
    `generator/2`, `nodeDeclaration/4`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `provrml/provrml_io`, `provrml/generator_util`, `provrml/parser_util`, `provrml/provrmlerror`, `provrml/internal_types`.

216.2 Documentation on exports (generator)

generator/2: PREDICATE

Usage: `generator(Terms,VRML)`

- *Description:* This predicate is the generator of VRML code. It accepts a list of terms that is correct VRML code, other kind of terms will be rejected will errormessage accordingly. The output is a string of correct VRML code, acceptable for VRML browsers.
- *Call and exit should be compatible with:*
  `Terms` is a list of `terms`. (basic_props:list/2)
  `VRML` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*
  `Terms` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `VRML` is a free variable. (term_typing:var/1)

nodeDeclaration/4: PREDICATE

Usage: `nodeDeclaration(ParseIn,ParseOut,L,T)`

- *Description:* The node declaration canbe constructed by a DEFinition, we then make a call to generator_util to make proper settings before continue. There can be a USE of a prior defined node or we can have a normal node declaration, one of the built ins.
- *Call and exit should be compatible with:*
  `ParseIn` is a parse structure. (internal_types:parse/1)
  `ParseOut` is a parse structure. (internal_types:parse/1)
  `L` is a list. (basic_props:list/1)
  `T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*
`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseOut` is a free variable. (term_typing:var/1)

217 generator_util (library)

Author(s): Göran Smedbäck.

217.1 Usage and interface (generator_util)

- **Library usage:**
  `:- use_module(library(generator_util)).`
- **Exports:**
  - *Predicates:*
    `reading/4`, `reading/5`, `reading/6`, `open_node/6`, `close_node/5`, `close_nodeGut/4`, `open_PROTO/4`, `close_PROTO/6`, `open_EXTERNPROTO/5`, `close_EXTERNPROTO/6`, `open_DEF/5`, `close_DEF/5`, `open_Script/5`, `close_Script/5`, `decompose_field/3`, `indentation_list/2`, `start_vrmlScene/4`, `remove_comments/4`.
- **Other modules used:**
  - *System library modules:*
    `provrml/provrmlerror`, `provrml/provrml_io`, `provrml/field_value`, `provrml/field_value_check`, `provrml/lookup`, `provrml/parser_util`.

217.2 Documentation on exports (generator_util)

reading/4: PREDICATE

Usage 1: `reading(DEF,Parse,L,T)`

- *Description:* This predicate will check if we have the special key word.
- *Call and exit should be compatible with:*

`DEF` is an atom.	(basic_props:atm/1)
`Parse` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

- *The following properties should hold at call time:*

`DEF` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Parse` is currently a term which is not a free variable.	(term_typing:nonvar/1)

Usage 2: `reading(IS,NodeTypeId,ParseIn,ParseOut)`

- *Description:* This predicate will refer to a formerly introduced interface. We do a checkup of the access type and output the values.
- *Call and exit should be compatible with:*

`IS` is an atom.	(basic_props:atm/1)
`NodeTypeId` is an atom.	(basic_props:atm/1)
`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)

- *The following properties should hold at call time:*

`IS` is currently a term which is not a free variable.	(term_typing:nonvar/1)

`NodeTypeId` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseOut` is a free variable. (term_typing:var/1)

Usage 3: `reading(Node,Parse,L,T)`

- *Description:* This predicate will read a node so we will check the properties of that one and then continue the progress in the generation.
- *Call and exit should be compatible with:*

`Node` is an atom. (basic_props:atm/1)
`Parse` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

- *The following properties should hold at call time:*

`Node` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 4: `reading(Script,Parse,L,T)`

- *Description:* This predicate read a script and will then continue the generation.
- *Call and exit should be compatible with:*

`Script` is an atom. (basic_props:atm/1)
`Parse` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

- *The following properties should hold at call time:*

`Script` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 5: `reading(NodeGut,NodeName,ParseIn,ParseOut)`

- *Description:* This predicate will read a node gut and will check the field according to the name.
- *Call and exit should be compatible with:*

`NodeGut` is an atom. (basic_props:atm/1)
`NodeName` is an atom. (basic_props:atm/1)
`ParseIn` is a parse structure. (internal_types:parse/1)
`ParseOut` is a parse structure. (internal_types:parse/1)

- *The following properties should hold at call time:*

`NodeGut` is currently a term which is not a free variable. (term_typing:nonvar/1)
`NodeName` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseOut` is a free variable. (term_typing:var/1)

Usage 6: `reading(PROTO,Parse,L,T)`

- *Description:* This predicate will read a prototype, check that the term name is the one, 'PROTO'.
- *Call and exit should be compatible with:*

`PROTO` is an atom. (basic_props:atm/1)
`Parse` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`PROTO` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

Usage 7: `reading(EXTERNPROTO,Parse,L,T)`

– *Description:* This predicate read a term with the name given as the first argument.

– *Call and exit should be compatible with:*

`EXTERNPROTO` is an atom. (basic_props:atm/1)

`Parse` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`EXTERNPROTO` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

reading/5: PREDICATE

Usage 1: `reading(Entrance,ParseIn,ParseOut,L,T)`

– *Description:* This predicate is a general predicate in this help module. The first argument is a key word to direct the input to the right entrance. When I say output in the following predicates I am refering to the out predicate which actually will use the features of DCG and we will add the output terms in a list for later output. The list is hidden to the user but is in this case the fourth argument will be the list and the fifth will be the resulting list, hopefully empty after the generation. Then we should have read all the terms.

– *Call and exit should be compatible with:*

`Entrance` is an atom. (basic_props:atm/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Entrance` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 2: `reading(Empty,ParseIn,ParseOut,L,T)`

– *Description:* This predicate check if we have ran out of input and is ready to terminate the reading.

– *Call and exit should be compatible with:*

`Empty` is an atom. (basic_props:atm/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Empty` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 3: `reading(Header,ParseIn,ParseOut,L,T)`

– *Description:* This predicate read the header and after the header we can have more information, that is comments.

– *Call and exit should be compatible with:*

`Header` is an atom. (basic_props:atm/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Header` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 4: `reading(NULL,ParseIn,ParseOut,L,T)`

– *Description:* This predicate accepts the special key word 'NULL' and will output that.

– *Call and exit should be compatible with:*

`NULL` is an atom. (basic_props:atm/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`NULL` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 5: `reading(Comment,ParseIn,ParseOut,L,T)`

– *Description:* This predicate will read a comment.

– *Call and exit should be compatible with:*

`Comment` is an atom. (basic_props:atm/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Comment` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 6: `reading(USE,ParseIn,ParseOut,L,T)`

– *Description:* This predicate will read the 'USE' key word.
– *Call and exit should be compatible with:*

`USE` is an atom. (basic_props:atm/1)
`ParseIn` is a parse structure. (internal_types:parse/1)
`ParseOut` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`USE` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseOut` is a free variable. (term_typing:var/1)

Usage 7: `reading(MfstringValue,ParseIn,ParseOut,L,T)`

– *Description:* This predicate will read the term multi field string value. Then it will continue the generation in the field_value_check module. There it will read the string values and generate the code.
– *Call and exit should be compatible with:*

`MfstringValue` is an atom. (basic_props:atm/1)
`ParseIn` is a parse structure. (internal_types:parse/1)
`ParseOut` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`MfstringValue` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseOut` is a free variable. (term_typing:var/1)

Usage 8: `reading(ExposedField,ParseIn,ParseOut,L,T)`

– *Description:* This predicate will read an exposedField and do the checkup for the interface with all its components.
– *Call and exit should be compatible with:*

`ExposedField` is an atom. (basic_props:atm/1)
`ParseIn` is a parse structure. (internal_types:parse/1)
`ParseOut` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`ExposedField` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseOut` is a free variable. (term_typing:var/1)

Usage 9: `reading(RestrictedInterfaceDeclaration,ParseIn,ParseOut,L,T)`

– *Description:* This predicate will read the declaration for a restricted field and do the checkup accordingly if necessary.

– *Call and exit should be compatible with:*

`RestrictedInterfaceDeclaration` is an atom. (basic_props:atm/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`RestrictedInterfaceDeclaration` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 10: `reading(ExternInterfaceDeclaration,ParseIn,ParseOut,L,T)`

– *Description:* For reading an external interface declaration we see that we have three arguments in the term and that we have special access key word. We then outputs the declaration.

– *Call and exit should be compatible with:*

`ExternInterfaceDeclaration` is an atom. (basic_props:atm/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`ExternInterfaceDeclaration` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 11: `reading(ROUTE,ParseIn,ParseOut,L,T)`

– *Description:* Reading a ROUTE and we split the term into its parts. There might be comments in the different fields and therefore we have to strip_clean the fields to get them without a possible list containing comments, this for the checkup of the routing parameters. We then do an output of the values, then with the comments.

– *Call and exit should be compatible with:*

`ROUTE` is an atom. (basic_props:atm/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`ROUTE` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 12: `reading(Error_X,ParseIn,ParseOut,L,T)`

– *Description:* The predicate will call a proper error message after fetching some values like name.

– *Call and exit should be compatible with:*

`Error_X` is a compound term.	(basic_props:struct/1)
`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

– *The following properties should hold at call time:*

`Error_X` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)

reading/6: PREDICATE

No further documentation available for this predicate.

open_node/6: PREDICATE

Usage: `open_node(ParseIn,ParseOut,NodeGutsParseStruct,NodeNameId,L,T)`

– *Description:* The predicate will open a node to extract its name and its guts. The guts will then be added to a new parse structure to be emptied in the above module, generator.

– *Call and exit should be compatible with:*

`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)
`NodeGutsParseStruct` is a parse structure.	(internal_types:parse/1)
`NodeNameId` is an atom.	(basic_props:atm/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)
`NodeGutsParseStruct` is a free variable.	(term_typing:var/1)
`NodeNameId` is a free variable.	(term_typing:var/1)

close_node/5: PREDICATE

Usage: `close_node(ParseNodeStruct,ParseIn,ParseOut,L,T)`

– *Description:* The predicate will end the generation from the node and will do that by adding all the new posts in the dictionary, like new declarations and nodes, to the already used dictionary.

– *Call and exit should be compatible with:*

`ParseNodeStruct` is a parse structure.	(internal_types:parse/1)
`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

– *The following properties should hold at call time:*

`ParseNodeStruct` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

close_nodeGut/4: PREDICATE

Usage: `close_nodeGut(ParseIn,ParseOut,L,T)`

– *Description:* The predicate will perform all the actions needed terminate the reading of the node guts.

– *Call and exit should be compatible with:*

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

open_PROTO/4: PREDICATE

Usage: `open_PROTO(Parse,ProtoParse,L,T)`

– *Description:* This predicate will construct a parse structure with the prototype information, the interface only. It can thereafter be used in further code generation. The scene will be opened afterwards.

– *Call and exit should be compatible with:*

`Parse` is a parse structure. (internal_types:parse/1)

`ProtoParse` is a parse structure. (internal_types:parse/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ProtoParse` is a free variable. (term_typing:var/1)

close_PROTO/6: PREDICATE

Usage: `close_PROTO(DeclParse,SceneParse,ParseIn,ParseOut,L,T)`

– *Description:* The predicate will push the dictionaries with its new information when gone through the scenery. The output parse structure will contain all the new information.

– *Call and exit should be compatible with:*

`DeclParse` is a parse structure. (internal_types:parse/1)

`SceneParse` is a parse structure. (internal_types:parse/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`SceneParse` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseOut` is a free variable. (term_typing:var/1)

open_EXTERNPROTO/5: PREDICATE

Usage: `open_EXTERNPROTO(Parse,DeclParse,StringParse,L,T)`

– *Description:* This predicate will construct parse structures with the prototype information, the interface and the strings. It can thereafter be used in further code generation.

– *Call and exit should be compatible with:*

`Parse` is a parse structure. (internal_types:parse/1)
`DeclParse` is a parse structure. (internal_types:parse/1)
`StringParse` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)
`DeclParse` is a free variable. (term_typing:var/1)
`StringParse` is a free variable. (term_typing:var/1)

close_EXTERNPROTO/6: PREDICATE

Usage: `close_EXTERNPROTO(ParseDeclIn,ParseStringIn,ParseIn,ParseOut,L,T)`

– *Description:* The predicate will end the generating of the external prototype and do checkup if there was correct.

– *Call and exit should be compatible with:*

`ParseDeclIn` is a parse structure. (internal_types:parse/1)
`ParseStringIn` is a parse structure. (internal_types:parse/1)
`ParseIn` is a parse structure. (internal_types:parse/1)
`ParseOut` is a parse structure. (internal_types:parse/1)
`L` is a list. (basic_props:list/1)
`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)
`ParseOut` is a free variable. (term_typing:var/1)

open_DEF/5: PREDICATE

Usage: `open_DEF(ParseIn,ParseOut,ParseNode,L,T)`

– *Description:* The predicate will open and do the settings to generate the code for a definition of a node.
– *Call and exit should be compatible with:*

`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)
`ParseNode` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)
`ParseNode` is a free variable.	(term_typing:var/1)

close_DEF/5: PREDICATE

Usage: `close_DEF(ParseNode,ParseIn,ParseOut,L,T)`

– *Description:* The predicate will push the new dictionary information from the node definition to the output parse structure combining the information in the old parse structure with the newly received.
– *Call and exit should be compatible with:*

`ParseNode` is a parse structure.	(internal_types:parse/1)
`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

– *The following properties should hold at call time:*

`ParseNode` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)

open_Script/5: PREDICATE

Usage: `open_Script(ParseIn,ParseOut,ScriptParse,L,T)`

– *Description:* The predicate will create a parse structure with the script guts.
– *Call and exit should be compatible with:*

`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)
`ScriptParse` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)
`ScriptParse` is a free variable.	(term_typing:var/1)

close_Script/5: PREDICATE

Usage: `close_Script(ScriptParse,ParseIn,ParseOut,L,T)`

- *Description:* This predicate will update the dictionaries after generating code for the script.
- *Call and exit should be compatible with:*

`ScriptParse` is a parse structure.	(internal_types:parse/1)
`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

- *The following properties should hold at call time:*

`ScriptParse` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)

decompose_field/3: PREDICATE

No further documentation available for this predicate.

indentation_list/2: PREDICATE

Usage: `indentation_list(Parse,IndList)`

- *Description:* This predcate will construct a list with indentations to be output before text. The information of the indentations is inside the parse structure.
- *Call and exit should be compatible with:*

`Parse` is a parse structure.	(internal_types:parse/1)
`IndList` is a list of `atm`s.	(basic_props:list/2)

- *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`IndList` is a free variable.	(term_typing:var/1)

start_vrmlScene/4: PREDICATE

Usage: `start_vrmlScene(Parse,ParseScene,L,T)`

- *Description:* The predicate will construct a parse structure with the prototype scene and do the setups.
- *Call and exit should be compatible with:*

`Parse` is a parse structure.	(internal_types:parse/1)
`ParseScene` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

- *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseScene` is a free variable.	(term_typing:var/1)

remove_comments/4: PREDICATE

Usage: `remove_comments(Value,CommentsBefore,ValueClean,CommentsAfter)`

- *Description:* The predicate will remove comments and return the comments before and after the pure value.
- *Call and exit should be compatible with:*

  `Value` is a list of `atm`s. (basic_props:list/2)

  `CommentsBefore` is a list of `atm`s. (basic_props:list/2)

  `ValueClean` is an atom. (basic_props:atm/1)

  `CommentsAfter` is a list of `atm`s. (basic_props:list/2)
- *The following properties should hold at call time:*

  `Value` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `CommentsBefore` is a free variable. (term_typing:var/1)

  `ValueClean` is a free variable. (term_typing:var/1)

  `CommentsAfter` is a free variable. (term_typing:var/1)

217.3 Known bugs and planned improvements (`generator_util`)

- Run-time checks have been reported not to work with this code. That means that either the assertions here, or the code that implements the run-time checks are erroneous.

218 internal_types (library)

Author(s): Göran Smedbäck.

These are the internal data types used in the predicates. They are only used to simplify this documentation and make it more understandable.

Implemented by Göran Smedbäck

218.1 Usage and interface (internal_types)

- **Library usage:**
  `:- use_module(library(internal_types)).`
- **Exports:**
  - *Regular Types:*
    `bound/1`, `bound_double/1`, `dictionary/1`, `environment/1`, `parse/1`, `tree/1`, `whitespace/1`.

218.2 Documentation on exports (internal_types)

bound/1: REGTYPE

Min is a number or an atom that indicates the minimal value, Max indicates the maximal.

```
bound(bound(Min,Max)) :-
        atm(Min),
        atm(Max).
```

Usage: `bound(Bound)`

- *Description:* `Bound` is a variable interval.

bound_double/1: REGTYPE

Min is a number or an atom that indicates the minimal value, Max indicates the maximal. The first two for some value and the second pair for some other. Typically used for types that are compound, e.g., rotationvalue.

```
bound_double(bound(Min0,Max0,Min1,Max1)) :-
        atm(Min0),
        atm(Max0),
        atm(Min1),
        atm(Max1).
```

Usage: `bound_double(Bound)`

- *Description:* `Bound` is a variable interval.

dictionary/1: REGTYPE

Dic is a tree structure and is used as the internal representation of the dictionary.

```
dictionary(dic(Dic)) :-
        tree(Dic).
dictionary(X) :-
        term(X).
```

Usage: `dictionary(Dictionary)`

– *Description:* `Dictionary` is a dictionary.

environment/1: REGTYPE

EnvironmentType one of 'DEF','PROTO','EXTERNPROTO' with the name Name. Whitespace is a structure with whitespace information.

```
environment(env(Env,Name,WhiteSpace)) :-
        atm(Env),
        atm(Name),
        whitespace(WhiteSpace).
```

Usage: `environment(Environment)`

– *Description:* `Environment` is an environment structure.

parse/1: REGTYPE

In is the list of tokens to parse and Out is the resulting list after the parsing. Env is of type env and is the environment-structure.The dictinonary Dic contains created information and structures.

```
parse(parse(In,Out,Env,Dic)) :-
        list(In),
        list(Out),
        environment(Env),
        dictionary(Dic).
```

Usage: `parse(Parse)`

– *Description:* `Parse` is a parse structure.

tree/1: REGTYPE

Key is the search-key, Leaf is the information, Left and Right are more dictionary posts, where Left have less Key-value.

```
tree(tree(Key,Leaf,Left,Right)) :-
        atm(Key),
        leaf(Leaf),
        tree(Left),
        tree(Right).
```

Usage: `tree(Tree)`

– *Description:* `Tree` is a tree structure.

whitespace/1: REGTYPE

The Row and Indentation information. The row information used when parsing the VRML code to give accurate error position and the indentation is used when generating VRML code from terms.

```
whitespace(w(Row,Indentation)) :-
        number(Row),
        number(Indentation).
```

Usage: `whitespace(Whitespace)`

– *Description:* `Whitespace` is a whitespace structure.

219 provrml_io (library)

Author(s): Göran Smedbäck.
This file implements I/O predicates of different types.
Implemented by Göran Smedbäck

219.1 Usage and interface (provrml_io)

- **Library usage:**
  `:- use_module(library(provrml_io)).`
- **Exports:**
  - *Predicates:*
    `out/1`, `out/3`, `convert_atoms_to_string/2`, `read_terms_file/2`, `write_terms_file/2`, `read_vrml_file/2`, `write_vrml_file/2`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `lists`.

219.2 Documentation on exports (provrml_io)

out/1: PREDICATE

Usage: `out(ListOfOutput)`

- *Description:* The predicate used is out/3 (DCG) where we will 'save' the output in the second argument. The tird argument is the rest, nil.
- *Call and exit should be compatible with:*
  `ListOfOutput` is a list of `atm`s. (basic_props:list/2)
- *The following properties should hold at call time:*
  `ListOfOutput` is currently a term which is not a free variable. (term_typing:nonvar/1)

out/3: PREDICATE

No further documentation available for this predicate.

convert_atoms_to_string/2: PREDICATE

Usage: `convert_atoms_to_string(Atoms,String)`

- *Description:* The predicate transforms a list of atoms to a string.
- *Call and exit should be compatible with:*
  `Atoms` is a list of `atm`s. (basic_props:list/2)
  `String` is a list of `num`s. (basic_props:list/2)
- *The following properties should hold at call time:*
  `Atoms` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `String` is a free variable. (term_typing:var/1)

read_terms_file/2: PREDICATE

Usage: `read_terms_file(Filename,Term)`

- *Description:* Given a filename to a file with terms, the predicate reads the terms and are returned in the second argument. `Filename` is an atom and `Term` is the read prolog terms.
- *Call and exit should be compatible with:*

  `Filename` is an atom. (basic_props:atm/1)

  `Term` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `Filename` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Term` is a free variable. (term_typing:var/1)

write_terms_file/2: PREDICATE

Usage: `write_terms_file(FileName,List)`

- *Description:* Given a filename and a list of terms the predicate will write them down to the file.
- *Call and exit should be compatible with:*

  `FileName` is an atom. (basic_props:atm/1)

  `List` is a list of `atm`s. (basic_props:list/2)
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `List` is currently a term which is not a free variable. (term_typing:nonvar/1)

read_vrml_file/2: PREDICATE

Usage: `read_vrml_file(FileName,Data)`

- *Description:* Given a filename, the predicate returns the substance.
- *Call and exit should be compatible with:*

  `FileName` is an atom. (basic_props:atm/1)

  `Data` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Data` is a free variable. (term_typing:var/1)

write_vrml_file/2: PREDICATE

Usage: `write_vrml_file(FileName,Data)`

- *Description:* Given a filename and data in form of a string, the predicate will write the data to the named file.
- *Call and exit should be compatible with:*

  `FileName` is an atom. (basic_props:atm/1)

  `Data` is a string (a list of character codes). (basic_props:string/1)
- *The following properties should hold at call time:*

  `FileName` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Data` is currently a term which is not a free variable. (term_typing:nonvar/1)

220 lookup (library)

Author(s): Göran Smedbäck.

220.1 Usage and interface (lookup)

- **Library usage:**
  `:- use_module(library(lookup)).`
- **Exports:**
  - *Predicates:*
    `create_proto_element/3`, `get_prototype_interface/2`, `get_prototype_definition/2`, `lookup_check_node/4`, `lookup_check_field/6`, `lookup_check_interface_fieldValue/8`, `lookup_field/4`, `lookup_route/5`, `lookup_fieldTypeId/1`, `lookup_get_fieldType/4`, `lookup_field_access/4`, `lookup_set_def/3`, `lookup_set_prototype/4`, `lookup_set_extern_prototype/4`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `provrml/provrmlerror`, `provrml/internal_types`, `provrml/provrml_io`, `provrml/parser_util`, `provrml/dictionary`, `provrml/dictionary_tree`, `provrml/field_value_check`, `provrml/boundary`, `provrml/generator_util`, `provrml/field_type`, `provrml/field_value`.

220.2 Documentation on exports (lookup)

create_proto_element/3: PREDICATE

Usage: `create_proto_element(Interface,Definition,Proto)`

- *Description:* The predicate will construct a proto structure containing the interface and the definition.
- *Call and exit should be compatible with:*

`Interface` is any term.	(basic_props:term/1)
`Definition` is any term.	(basic_props:term/1)
`Proto` is any term.	(basic_props:term/1)

- *The following properties should hold at call time:*

`Interface` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Definition` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Proto` is a free variable.	(term_typing:var/1)

get_prototype_interface/2: PREDICATE

Usage: `get_prototype_interface(Proto,Interface)`

- *Description:* The predicate will return the interface from a proto structure.
- *Call and exit should be compatible with:*

`Proto` is any term.	(basic_props:term/1)
`Interface` is any term.	(basic_props:term/1)

– *The following properties should hold at call time:*

`Proto` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Interface` is a free variable. (term_typing:var/1)

get_prototype_definition/2: PREDICATE

Usage: `get_prototype_definition(Proto,Definition)`

– *Description:* The predicate will return the definition from a proto structure.

– *Call and exit should be compatible with:*

`Proto` is any term. (basic_props:term/1)

`Definition` is any term. (basic_props:term/1)

– *The following properties should hold at call time:*

`Proto` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Definition` is a free variable. (term_typing:var/1)

lookup_check_node/4: PREDICATE

Usage: `lookup_check_node(ParseIn,NodeTypeId,L,T)`

– *Description:* The predicate will check so that the node is of an acceptable type. If the node name is not found in the ordinary dictionary then the secondary dictionary is consulted, the personal one. Then the node have to be a Prototype, Externproto or a Defined one.

– *Call and exit should be compatible with:*

field_value:parse(ParseIn) (field_value:parse/1)

`NodeTypeId` is an atom. (basic_props:atm/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`NodeTypeId` is currently a term which is not a free variable. (term_typing:nonvar/1)

lookup_check_field/6: PREDICATE

Usage: `lookup_check_field(ParseIn,ParseOut,NodeTypeId,Field,L,T)`

– *Description:* The predicate will create some output through the DCG and the output command out/3. There will be formatting and the most important part there will be a check of the field type and of its values so that they correspond to the type.

– *Call and exit should be compatible with:*

field_value:parse(ParseIn) (field_value:parse/1)

field_value:parse(ParseOut) (field_value:parse/1)

`NodeTypeId` is an atom. (basic_props:atm/1)

`Field` is any term. (basic_props:term/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)
`NodeTypeId` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Field` is currently a term which is not a free variable.	(term_typing:nonvar/1)

lookup_check_interface_fieldValue/8: PREDICATE

Usage: `lookup_check_interface_fieldValue(ParseIn,ParseOut,AccessType,FieldType,Id,FieldValue,DCGIn,DCGOut)`

– *Description:* The predicate formats the output for the interface part of the prototype. It also checks the values for the fields.

– *Call and exit should be compatible with:*

field_value:parse(ParseIn)	(field_value:parse/1)
field_value:parse(ParseOut)	(field_value:parse/1)
`AccessType` is an atom.	(basic_props:atm/1)
`FieldType` is any term.	(basic_props:term/1)
`Id` is an atom.	(basic_props:atm/1)
`FieldValue` is any term.	(basic_props:term/1)
`DCGIn` is a string (a list of character codes).	(basic_props:string/1)
`DCGOut` is a string (a list of character codes).	(basic_props:string/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)
`AccessType` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldType` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Id` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldValue` is currently a term which is not a free variable.	(term_typing:nonvar/1)

lookup_field/4: PREDICATE

Usage: `lookup_field(Parse,FieldTypeId,FieldId0,FieldId1)`

– *Description:* The predicate will control that the two connected Fields are of the same type, e.g., SFColor - SFColor, MFVec3f - MFVec3f.

– *Call and exit should be compatible with:*

field_value:parse(Parse)	(field_value:parse/1)
`FieldTypeId` is an atom.	(basic_props:atm/1)
`FieldId0` is an atom.	(basic_props:atm/1)
`FieldId1` is an atom.	(basic_props:atm/1)

– *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldTypeId` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldId0` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldId1` is currently a term which is not a free variable.	(term_typing:nonvar/1)

lookup_route/5: PREDICATE

Usage: `lookup_route(Parse,NodeTypeId0,FieldId0,NodeTypeId1,FieldId1)`

- *Description:* The predicate will check the routing behaviour for two given fields. They will be checked according to the binding rules, like name changes access proporties. The node types for the field must of course be given for the identification.
- *Call and exit should be compatible with:*

  field_value:parse(Parse) (field_value:parse/1)

  `NodeTypeId0` is an atom. (basic_props:atm/1)

  `FieldId0` is an atom. (basic_props:atm/1)

  `NodeTypeId1` is an atom. (basic_props:atm/1)

  `FieldId1` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `NodeTypeId0` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FieldId0` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `NodeTypeId1` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FieldId1` is currently a term which is not a free variable. (term_typing:nonvar/1)

lookup_fieldTypeId/1: PREDICATE

Usage: `lookup_fieldTypeId(FieldTypeId)`

- *Description:* The predicate just make a check to see if the given FieldType id is among the allowed. You can not construct own ones and the check is mearly a spellcheck.
- *Call and exit should be compatible with:*

  `FieldTypeId` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `FieldTypeId` is currently a term which is not a free variable. (term_typing:nonvar/1)

lookup_get_fieldType/4: PREDICATE

Usage: `lookup_get_fieldType(Parse,NodeTypeId,FieldId,FieldType)`

- *Description:* The predicate will return the given field's type. It will start the search in the ordinar dictionary and then to the personal dictionary sarting off with 'PROTO'. After it will go for 'DEF' and 'EXTERNPROTO'.
- *Call and exit should be compatible with:*

  field_value:parse(Parse) (field_value:parse/1)

  `NodeTypeId` is an atom. (basic_props:atm/1)

  `FieldId` is an atom. (basic_props:atm/1)

  `FieldType` is an atom. (basic_props:atm/1)
- *The following properties should hold at call time:*

  `Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `NodeTypeId` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FieldId` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FieldType` is a free variable. (term_typing:var/1)

lookup_field_access/4: PREDICATE

Usage: `lookup_field_access(Parse,NodenameId,FieldId,FieldId)`

- *Description:* The predicate will control that the access properties are correct according to the certain rules that we have. It makes a check to see if the fields are of the same access type or if one of them is an exposedField. It is not doing a route check up to control that behaviour entirely.
- *Call and exit should be compatible with:*

field_value:parse(Parse)	(field_value:parse/1)
`NodenameId` is an atom.	(basic_props:atm/1)
`FieldId` is an atom.	(basic_props:atm/1)
`FieldId` is an atom.	(basic_props:atm/1)

- *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`NodenameId` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldId` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldId` is currently a term which is not a free variable.	(term_typing:nonvar/1)

lookup_set_def/3: PREDICATE

Usage: `lookup_set_def(Parse,Name,Node)`

- *Description:* The predicate will enter a new post in the personal dictionary for the node definition.
- *Call and exit should be compatible with:*

field_value:parse(Parse)	(field_value:parse/1)
`Name` is an atom.	(basic_props:atm/1)
`Node` is any term.	(basic_props:term/1)

- *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Name` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Node` is currently a term which is not a free variable.	(term_typing:nonvar/1)

lookup_set_prototype/4: PREDICATE

Usage: `lookup_set_prototype(Parse,Name,Interface,Definition)`

- *Description:* The predicate will insert the prototype definition in the personal dictionary and will give a warning if there is a multiple name given.
- *Call and exit should be compatible with:*

field_value:parse(Parse)	(field_value:parse/1)
`Name` is an atom.	(basic_props:atm/1)
`Interface` is any term.	(basic_props:term/1)
`Definition` is any term.	(basic_props:term/1)

- *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Name` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Interface` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Definition` is currently a term which is not a free variable.	(term_typing:nonvar/1)

lookup_set_extern_prototype/4: PREDICATE

Usage: `lookup_set_extern_prototype(Parse,Name,Interface,Strings)`

- *Description:* The predicate will insert the external prototype definition in the personal dictionary and will give a warning if there is a multiple name given.
- *Call and exit should be compatible with:*

field_value:parse(Parse)	(field_value:parse/1)
`Name` is an atom.	(basic_props:atm/1)
`Interface` is any term.	(basic_props:term/1)
`Strings` is any term.	(basic_props:term/1)

- *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Name` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Interface` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Strings` is currently a term which is not a free variable.	(term_typing:nonvar/1)

221 provrml_parser (library)

Author(s): Göran Smedbäck.

221.1 Usage and interface (provrml_parser)

- **Library usage:**
  `:- use_module(library(provrml_parser)).`
- **Exports:**
  - *Predicates:*
    `parser/2`, `nodeDeclaration/4`.
  - *Properties:*
    `field_Id/1`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `lists`, `provrml/lookup`, `provrml/field_value`, `provrml/tokeniser`, `provrml/parser_util`, `provrml/possible`, `provrml/provrmlerror`.

221.2 Documentation on exports (provrml_parser)

parser/2: PREDICATE

Usage: `parser(VRML,Terms)`

- *Description:* The parser uses a tokeniser to read the input text string of VRML code and returns a list with the corresponding terms. The tokens will be read in this parser as the grammar says. The parser is according to the specification of the VRML grammar, accept that it is performed over tokens in sted of the actual code.
- *Call and exit should be compatible with:*
  `VRML` is a string (a list of character codes). (basic_props:string/1)
  `Terms` is a list of `terms`. (basic_props:list/2)
- *The following properties should hold at call time:*
  `VRML` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Terms` is a free variable. (term_typing:var/1)

nodeDeclaration/4: PREDICATE

Usage: `nodeDeclaration(Parse_in,Parse_out,L,T)`

- *Description:* The predicate is also accepted as a node field as has to be accessed from the module that reads field values, i.e., field_value.pl
- *Call and exit should be compatible with:*
  `Parse_in` is a parse structure. (internal_types:parse/1)
  `Parse_out` is a parse structure. (internal_types:parse/1)
  `L` is a list. (basic_props:list/1)
  `T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*
`Parse_in` is currently a term which is not a free variable. (term_typing:nonvar/1)
`Parse_out` is a free variable. (term_typing:var/1)

field_Id/1: PROPERTY

A property, defined as follows:

```
field_Id(_1).
```

222 parser_util (library)

Author(s): Göran Smedbäck.

222.1 Usage and interface (parser_util)

- **Library usage:**
  `:- use_module(library(parser_util)).`
- **Exports:**
  - *Predicates:*
    `at_least_one/4`, `at_least_one/5`, `fillout/4`, `fillout/5`, `create_node/3`, `create_field/3`, `create_field/4`, `create_field/5`, `create_directed_field/5`, `correct_commenting/4`, `create_parse_structure/1`, `create_parse_structure/2`, `create_parse_structure/3`, `create_environment/4`, `insert_comments_in_beginning/3`, `get_environment_name/2`, `get_environment_type/2`, `get_row_number/2`, `add_environment_whitespace/3`, `get_indentation/2`, `inc_indentation/2`, `dec_indentation/2`, `add_indentation/3`, `reduce_indentation/3`, `push_whitespace/3`, `push_dictionaries/3`, `get_parsed/2`, `get_environment/2`, `inside_proto/1`, `get_dictionaries/2`, `strip_from_list/2`, `strip_from_term/2`, `strip_clean/2`, `strip_exposed/2`, `strip_restricted/2`, `strip_interface/2`, `set_parsed/3`, `set_environment/3`, `insert_parsed/3`, `reverse_parsed/2`, `stop_parse/2`, `look_first_parsed/2`, `get_first_parsed/3`, `remove_code/3`, `look_ahead/3`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `dynamic`, `iso_misc`, `iso_byte_char`, `iso_incomplete`, `operators`, `read`, `write`, `lists`, `provrml/dictionary_tree`, `provrml/internal_types`.

222.2 Documentation on exports (parser_util)

at_least_one/4: PREDICATE

Usage: `at_least_one(ParseIn,ParseOut,L,T)`

- *Description:* One or more whitespace or comment have to be read, for the moment there are no whitespaces to be read so we only stick with the comments.
- *Call and exit should be compatible with:*

`ParseIn` is a parse structure.	(internal_types:parse/1)
`ParseOut` is a parse structure.	(internal_types:parse/1)
`L` is a list.	(basic_props:list/1)
`T` is a list.	(basic_props:list/1)

- *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`ParseOut` is a free variable.	(term_typing:var/1)

at_least_one/5: PREDICATE

Usage: `at_least_one(ParseIn,ParseOut,ListOfRead,L,T)`

- *Description:* One or more whitespace or comment have to be read, the result will be given in the list.
- *Call and exit should be compatible with:*

  `ParseIn` is a parse structure. (internal_types:parse/1)

  `ParseOut` is a parse structure. (internal_types:parse/1)

  `ListOfRead` is a list of `term`s. (basic_props:list/2)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*

  `ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseOut` is a free variable. (term_typing:var/1)

  `ListOfRead` is a free variable. (term_typing:var/1)

fillout/4: PREDICATE

Usage: `fillout(ParseIn,ParseOut,L,T)`

- *Description:* If there are whitespaces and comments, zero or more of each. This read all comments and all whitespace. The comments and whitespace will not be returned.
- *Call and exit should be compatible with:*

  `ParseIn` is a parse structure. (internal_types:parse/1)

  `ParseOut` is a parse structure. (internal_types:parse/1)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*

  `ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseOut` is a free variable. (term_typing:var/1)

fillout/5: PREDICATE

Usage: `fillout(ParseIn,ParseOut,ResultingList,L,T)`

- *Description:* If there are whitespaces and comments, zero or more of each, we add them to the resulting list. This read all comments and all whitespace.
- *Call and exit should be compatible with:*

  `ParseIn` is a parse structure. (internal_types:parse/1)

  `ParseOut` is a parse structure. (internal_types:parse/1)

  `ResultingList` is a list of `term`s. (basic_props:list/2)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*

  `ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseOut` is a free variable. (term_typing:var/1)

  `ResultingList` is a free variable. (term_typing:var/1)

create_node/3: PREDICATE

Usage: `create_node(NodeTypeId,Parse,Node)`

- *Description:* The predicate will construct a node term with the read guts which is inside the parse structure. A node consists of its name and one argument, a list of its fields.
- *Call and exit should be compatible with:*

  `NodeTypeId` is an atom. (basic_props:atm/1)

  `Parse` is a parse structure. (internal_types:parse/1)

  `Node` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*

  `NodeTypeId` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Node` is a free variable. (term_typing:var/1)

create_field/3: PREDICATE

Usage: `create_field(FieldNameId,Arguments,Field)`

- *Description:* The predicate will construct a field with the Id as the fieldname and the arguments as they are, in a double list, which results in a single list or a single list which will result in free arguments.
- *Call and exit should be compatible with:*

  `FieldNameId` is an atom. (basic_props:atm/1)

  `Arguments` is any term. (basic_props:term/1)

  `Field` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*

  `FieldNameId` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Arguments` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Field` is a free variable. (term_typing:var/1)

create_field/4: PREDICATE

Usage: `create_field(FieldAccess,FieldType,FieldId,Field)`

- *Description:* The predicate will construct a field with its access type as the name with type and id as arguments.
- *Call and exit should be compatible with:*

  `FieldAccess` is an atom. (basic_props:atm/1)

  `FieldType` is an atom. (basic_props:atm/1)

  `FieldId` is an atom. (basic_props:atm/1)

  `Field` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*

  `FieldAccess` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FieldType` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `FieldId` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Field` is a free variable. (term_typing:var/1)

create_field/5: PREDICATE

Usage: `create_field(FieldAccess,FieldType,FieldId,Fieldvalue,Field)`

- *Description:* The predicate will construct a field with its access type as the name with type, id and value as arguments.
- *Call and exit should be compatible with:*

`FieldAccess` is an atom.	(basic_props:atm/1)
`FieldType` is an atom.	(basic_props:atm/1)
`FieldId` is an atom.	(basic_props:atm/1)
`Fieldvalue` is any term.	(basic_props:term/1)
`Field` is any term.	(basic_props:term/1)

- *The following properties should hold at call time:*

`FieldAccess` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldType` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`FieldId` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Fieldvalue` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Field` is a free variable.	(term_typing:var/1)

create_directed_field/5: PREDICATE

Usage: `create_directed_field(Access,Type,Id0,Id1,Field)`

- *Description:* The predicate will construct a directed field with the key word IS in the middle. Its access type as the name with type, from id0 and to id1 as arguments.
- *Call and exit should be compatible with:*

`Access` is an atom.	(basic_props:atm/1)
`Type` is an atom.	(basic_props:atm/1)
`Id0` is an atom.	(basic_props:atm/1)
`Id1` is an atom.	(basic_props:atm/1)
`Field` is any term.	(basic_props:term/1)

- *The following properties should hold at call time:*

`Access` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Type` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Id0` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Id1` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`Field` is a free variable.	(term_typing:var/1)

correct_commenting/4: PREDICATE

Usage: `correct_commenting(Place,Comment,ParsedIn,ParsedOut)`

- *Description:* The predicate places the comment 'before' or 'after' the parsed term. This results in a list with the term and the comment or in just returning the term.
- *Call and exit should be compatible with:*

`Place` is an atom.	(basic_props:atm/1)
`Comment` is a compound term.	(basic_props:struct/1)
`ParsedIn` is any term.	(basic_props:term/1)
`ParsedOut` is any term.	(basic_props:term/1)

– *The following properties should hold at call time:*

`Place` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Comment` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParsedIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParsedOut` is a free variable. (term_typing:var/1)

create_parse_structure/1: PREDICATE

Usage: `create_parse_structure(Parse)`

– *Description:* The predicate will construct the parse structure with its three fields: the parsing list, the environment structure, and the dictionaries.

– *Call and exit should be compatible with:*

`Parse` is a parse structure. (internal_types:parse/1)

– *The following properties should hold at call time:*

`Parse` is a free variable. (term_typing:var/1)

create_parse_structure/2: PREDICATE

Usage 1: `create_parse_structure(ParseIn,ParseOut)`

– *Description:* The predicate will construct a parse structure with its three fields: the parsing list, the environment structure, and the dictionaries. It will reuse the environment and the dictionaries from the input.

– *Call and exit should be compatible with:*

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

Usage 2: `create_parse_structure(ParsedList,ParseOut)`

– *Description:* The predicate will construct a parse structure with its three fields: the parsing list, the environment structure, and the dictionaries. It will use the list of parsed items in its structure.

– *Call and exit should be compatible with:*

`ParsedList` is a list of `terms`. (basic_props:list/2)

`ParseOut` is a parse structure. (internal_types:parse/1)

– *The following properties should hold at call time:*

`ParsedList` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

create_parse_structure/3: PREDICATE

Usage: `create_parse_structure(ParsedList,ParseIn,ParseOut)`

– *Description:* The predicate will construct a parse structure with its three fields: the parsing list, the environment structure, and the dictionaries. It will use the list of parsed items in its structure and the environment and the dictionary from the parse structure given.

– *Call and exit should be compatible with:*

`ParsedList` is a list of `terms`. (basic_props:list/2)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

– *The following properties should hold at call time:*

`ParsedList` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

create_environment/4: PREDICATE

Usage: `create_environment(Parse,EnvType,Name,EnvStruct)`

– *Description:* The predicate will construct an environment structure based on the information in the parse structure. Well only the white- space information will be reused. The are three types of environments 'PROTO', 'EXTERNPROTO', and 'DEF'.

– *Call and exit should be compatible with:*

`Parse` is a parse structure. (internal_types:parse/1)

`EnvType` is an atom. (basic_props:atm/1)

`Name` is an atom. (basic_props:atm/1)

`EnvStruct` is an environment structure. (internal_types:environment/1)

– *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

`EnvType` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Name` is currently a term which is not a free variable. (term_typing:nonvar/1)

`EnvStruct` is a free variable. (term_typing:var/1)

insert_comments_in_beginning/3: PREDICATE

Usage: `insert_comments_in_beginning(Comment,ParseIn,ParseOut)`

– *Description:* We add the comment in the beginneing of the parsed, to get the proper look.

– *Call and exit should be compatible with:*

`Comment` is a compound term. (basic_props:struct/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

– *The following properties should hold at call time:*

`Comment` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

get_environment_name/2: PREDICATE

Usage: `get_environment_name(Environment,Name)`

– *Description:* The predicate will return the enviroment name.

– *Call and exit should be compatible with:*

`Environment` is an environment structure. (internal_types:environment/1)

`Name` is an atom. (basic_props:atm/1)

– *The following properties should hold at call time:*

`Environment` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Name` is a free variable. (term_typing:var/1)

get_environment_type/2: PREDICATE

Usage: `get_environment_type(Environment,Type)`

– *Description:* The predicate will return the enviroment type, one of the three: 'PROTO', 'EXTERNPROTO', and 'DEF'.

– *Call and exit should be compatible with:*

`Environment` is an environment structure. (internal_types:environment/1)

`Type` is an atom. (basic_props:atm/1)

– *The following properties should hold at call time:*

`Environment` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Type` is a free variable. (term_typing:var/1)

get_row_number/2: PREDICATE

Usage: `get_row_number(Parse,Row)`

– *Description:* The predicate will return the row number from the parse structure. The row number is not fully implemented.

– *Call and exit should be compatible with:*

`Parse` is a parse structure. (internal_types:parse/1)

`Row` is a number. (basic_props:num/1)

– *The following properties should hold at call time:*

`Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Row` is a free variable. (term_typing:var/1)

add_environment_whitespace/3: PREDICATE

Usage: `add_environment_whitespace(EnvIn,WhiteSpaceList,EnvOut)`

– *Description:* The predicate will add the new whitespace that is collected in a list of whitespaces to the environment.

– *Call and exit should be compatible with:*

`EnvIn` is an environment structure. (internal_types:environment/1)

`WhiteSpaceList` is a list of `atm`s. (basic_props:list/2)

`EnvOut` is an environment structure. (internal_types:environment/1)

– *The following properties should hold at call time:*

`EnvIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`WhiteSpaceList` is currently a term which is not a free variable. (term_typing:nonvar/1)

`EnvOut` is a free variable. (term_typing:var/1)

get_indentation/2: PREDICATE

Usage 1: `get_indentation(Whitespace,Indentation)`

- *Description:* The predicate will return the indentation depth from a whitespace structure.
- *Call and exit should be compatible with:*

  `Whitespace` is a whitespace structure. (internal_types:whitespace/1)

  `Indentation` is a number. (basic_props:num/1)
- *The following properties should hold at call time:*

  `Whitespace` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Indentation` is a free variable. (term_typing:var/1)

Usage 2: `get_indentation(Parse,Indentation)`

- *Description:* The predicate will return the indentation depth from a parse structure.
- *Call and exit should be compatible with:*

  `Parse` is a parse structure. (internal_types:parse/1)

  `Indentation` is a number. (basic_props:num/1)
- *The following properties should hold at call time:*

  `Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Indentation` is a free variable. (term_typing:var/1)

inc_indentation/2: PREDICATE

Usage: `inc_indentation(ParseIn,ParseOut)`

- *Description:* Will increase the indentation with one step to a parse structure.
- *Call and exit should be compatible with:*

  `ParseIn` is a parse structure. (internal_types:parse/1)

  `ParseOut` is a parse structure. (internal_types:parse/1)
- *The following properties should hold at call time:*

  `ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseOut` is a free variable. (term_typing:var/1)

dec_indentation/2: PREDICATE

Usage: `dec_indentation(ParseIn,ParseOut)`

- *Description:* Will decrease the indentation with one step to a parse structure.
- *Call and exit should be compatible with:*

  `ParseIn` is a parse structure. (internal_types:parse/1)

  `ParseOut` is a parse structure. (internal_types:parse/1)
- *The following properties should hold at call time:*

  `ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseOut` is a free variable. (term_typing:var/1)

add_indentation/3: PREDICATE

No further documentation available for this predicate.

reduce_indentation/3: PREDICATE

No further documentation available for this predicate.

push_whitespace/3: PREDICATE

Usage: `push_whitespace(ParseWithWhitespace,ParseIn,ParseOut)`

- *Description:* The predicate will add the whitespace values from one parse structure to another one, resultin in the output, with the values from the second parse structure with the whitespace from the first added.
- *Call and exit should be compatible with:*

  `ParseWithWhitespace` is a parse structure. (internal_types:parse/1)

  `ParseIn` is a parse structure. (internal_types:parse/1)

  `ParseOut` is a parse structure. (internal_types:parse/1)
- *The following properties should hold at call time:*

  `ParseWithWhitespace` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseOut` is a free variable. (term_typing:var/1)

push_dictionaries/3: PREDICATE

Usage: `push_dictionaries(Parse,Parse,Parse)`

- *Description:* The predicate will set the first parse structure's directory to the second parsing structure argument. The resulting parsing structure will be returned.
- *Call and exit should be compatible with:*

  `Parse` is a parse structure. (internal_types:parse/1)

  `Parse` is a parse structure. (internal_types:parse/1)

  `Parse` is a parse structure. (internal_types:parse/1)
- *The following properties should hold at call time:*

  `Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Parse` is a free variable. (term_typing:var/1)

get_parsed/2: PREDICATE

Usage 1: `get_parsed(ParseStructure,ListOfParsed)`

- *Description:* The predicate will return a list of the parsed terms that is inside the parse structure.
- *Call and exit should be compatible with:*

  `ParseStructure` is a parse structure. (internal_types:parse/1)

  `ListOfParsed` is a list of `term`s. (basic_props:list/2)
- *The following properties should hold at call time:*

  `ParseStructure` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ListOfParsed` is a free variable. (term_typing:var/1)

Usage 2: `get_parsed(ParseStructure,EnvironmentStructure)`

- *Description:* The predicate will return the environment of the parse structure.
- *Call and exit should be compatible with:*

  `ParseStructure` is a parse structure. (internal_types:parse/1)

  `EnvironmentStructure` is an environment structure. (internal_types:environment/1)
- *The following properties should hold at call time:*

  `ParseStructure` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `EnvironmentStructure` is a free variable. (term_typing:var/1)

Usage 3: `get_parsed(ParseStructure,Dictionaries)`

- *Description:* The predicate will return dictionary used within the parse structure.
- *Call and exit should be compatible with:*

  `ParseStructure` is a parse structure. (internal_types:parse/1)

  `Dictionaries` is a dictionary. (internal_types:dictionary/1)
- *The following properties should hold at call time:*

  `ParseStructure` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Dictionaries` is a free variable. (term_typing:var/1)

get_environment/2: PREDICATE

No further documentation available for this predicate.

inside_proto/1: PREDICATE

Usage: `inside_proto(Parse)`

- *Description:* The predicate will answer to the question: are we parsing inside a PROTO/EXTERNPROTO.
- *Call and exit should be compatible with:*

  `Parse` is a parse structure. (internal_types:parse/1)
- *The following properties should hold at call time:*

  `Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

get_dictionaries/2: PREDICATE

No further documentation available for this predicate.

strip_from_list/2: PREDICATE

Usage: `strip_from_list(ListWithComments,CleanList)`

- *Description:* The predicate will strip the list from comments and return a list without any comments.
- *Call and exit should be compatible with:*

  `ListWithComments` is a list of `terms`. (basic_props:list/2)

  `CleanList` is a list of `terms`. (basic_props:list/2)

– *The following properties should hold at call time:*

`ListWithComments` is currently a term which is not a free variable. (term_typing:nonvar/1)

`CleanList` is a free variable. (term_typing:var/1)

strip_from_term/2: PREDICATE

Usage: `strip_from_term(Term,Stripped)`

– *Description:* The predicate will remove comments from a term, it will reduce its arguments if there are comments as arguments.

– *Call and exit should be compatible with:*

`Term` is any term. (basic_props:term/1)

`Stripped` is any term. (basic_props:term/1)

– *The following properties should hold at call time:*

`Term` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Stripped` is a free variable. (term_typing:var/1)

strip_clean/2: PREDICATE

Usage: `strip_clean(ParsedIn,ParsedOut)`

– *Description:* The predicate will return a striped list or a single atom if there was no comments and no more items in the list. It will also return a atom if there is comments and only one other element.

– *Call and exit should be compatible with:*

`ParsedIn` is any term. (basic_props:term/1)

`ParsedOut` is any term. (basic_props:term/1)

– *The following properties should hold at call time:*

`ParsedIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParsedOut` is a free variable. (term_typing:var/1)

strip_exposed/2: PREDICATE

No further documentation available for this predicate.

strip_restricted/2: PREDICATE

No further documentation available for this predicate.

strip_interface/2: PREDICATE

Usage: `strip_interface(Interface,StrippedInterface)`

– *Description:* The predicate will remove comments from the interface that we read for the PROTOtype. This will help us when setting the properties.

– *Call and exit should be compatible with:*

`Interface` is a list of `terms`. (basic_props:list/2)

`StrippedInterface` is a list of `terms`. (basic_props:list/2)

– *The following properties should hold at call time:*

`Interface` is currently a term which is not a free variable. (term_typing:nonvar/1)

`StrippedInterface` is a free variable. (term_typing:var/1)

set_parsed/3: PREDICATE

Usage: `set_parsed(ParseIn,NewParseList,ParseOut)`

– *Description:* The predicate will create a new parse structure from the first parse structure with the parse list from the second argument.

– *Call and exit should be compatible with:*

`ParseIn` is a parse structure. (internal_types:parse/1)

`NewParseList` is a list of `term`s. (basic_props:list/2)

`ParseOut` is a parse structure. (internal_types:parse/1)

– *The following properties should hold at call time:*

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`NewParseList` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

set_environment/3: PREDICATE

Usage: `set_environment(Environment,ParseIn,ParseOut)`

– *Description:* The modificator will return a parse structure with the environment given with the other properties from the first parse structure.

– *Call and exit should be compatible with:*

`Environment` is an environment structure. (internal_types:environment/1)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

– *The following properties should hold at call time:*

`Environment` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

insert_parsed/3: PREDICATE

Usage: `insert_parsed(ListOfParsed,ParseIn,ParseOut)`

– *Description:* The predicate will append the newly parsed terms to the old that we have in the parse structure.

– *Call and exit should be compatible with:*

`ListOfParsed` is a list of `term`s. (basic_props:list/2)

`ParseIn` is a parse structure. (internal_types:parse/1)

`ParseOut` is a parse structure. (internal_types:parse/1)

– *The following properties should hold at call time:*

`ListOfParsed` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

`ParseOut` is a free variable. (term_typing:var/1)

reverse_parsed/2: PREDICATE

Usage: `reverse_parsed(ParseIn,ParseOut)`

- *Description:* The returned parse structure is the same as the input with the difference that the parsed terms are in reverse order.
- *Call and exit should be compatible with:*

  `ParseIn` is a parse structure. (internal_types:parse/1)

  `ParseOut` is a parse structure. (internal_types:parse/1)
- *The following properties should hold at call time:*

  `ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseOut` is a free variable. (term_typing:var/1)

stop_parse/2: PREDICATE

Usage: `stop_parse(TermIn,TermOut)`

- *Description:* The predicate will bind the invalue to the outvalue, used to terminate a parsing.
- *Call and exit should be compatible with:*

  `TermIn` is any term. (basic_props:term/1)

  `TermOut` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*

  `TermIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `TermOut` is a free variable. (term_typing:var/1)

look_first_parsed/2: PREDICATE

Usage: `look_first_parsed(Parse,First)`

- *Description:* Look at the first item in the parse structure.
- *Call and exit should be compatible with:*

  `Parse` is a parse structure. (internal_types:parse/1)

  `First` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*

  `Parse` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `First` is a free variable. (term_typing:var/1)

get_first_parsed/3: PREDICATE

Usage: `get_first_parsed(ParseIn,ParseOut,First)`

- *Description:* Get the first item in the parse structure and return the parse structure with the item removed.
- *Call and exit should be compatible with:*

  `ParseIn` is a parse structure. (internal_types:parse/1)

  `ParseOut` is a parse structure. (internal_types:parse/1)

  `First` is any term. (basic_props:term/1)
- *The following properties should hold at call time:*

  `ParseIn` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `ParseOut` is a free variable. (term_typing:var/1)

  `First` is a free variable. (term_typing:var/1)

remove_code/3: PREDICATE

No further documentation available for this predicate.

look_ahead/3: PREDICATE

Usage: `look_ahead(Name,Parsed,Parsed)`

- *Description:* This predicate is used normally by the CDG and the two last arguments will therefore be the same because we don't remove the parsed. The name is the name inside a term, the first argument.
- *Call and exit should be compatible with:*

  `Name` is an atom. (basic_props:atm/1)

  `Parsed` is a list of `term`s. (basic_props:list/2)

  `Parsed` is a list of `term`s. (basic_props:list/2)
- *The following properties should hold at call time:*

  `Name` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Parsed` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Parsed` is a free variable. (term_typing:var/1)

223 possible (library)

Author(s): Göran Smedbäck.

223.1 Usage and interface (possible)

- **Library usage:**
  `:- use_module(library(possible)).`
- **Exports:**
  - *Predicates:*
    `continue/3`.

223.2 Documentation on exports (possible)

continue/3: PREDICATE

Usage: `continue(RuleName,L,T)`

- *Description:* The predicate will check the rule name's possible followers, that is the entrance in the grammar, to see if we have the possibility to contiue the generation in the asked direction. We will by using the DCG look one item ahead to see if we have the proper key name ahead. Then we check the possible alternatives for the rule name from the list and type we receive from the call to possible. As we can see there are different look ahead predicates depending if we have a token or a word, looking on the structure name(token name) or if we should read the first word beyond the structure name.
- *Call and exit should be compatible with:*
  `RuleName` is an atom. (basic_props:atm/1)
  `L` is a list. (basic_props:list/1)
  `T` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*
  `RuleName` is currently a term which is not a free variable. (term_typing:nonvar/1)

224 tokeniser (library)

Author(s): Göran Smedbäck.

224.1 Usage and interface (tokeniser)

- **Library usage:**
  `:- use_module(library(tokeniser)).`
- **Exports:**
  - *Predicates:*
    `tokeniser/2`, `token_read/3`.
- **Other modules used:**
  - *System library modules:*
    `lists`, `provrml/provrmlerror`.

224.2 Documentation on exports (tokeniser)

tokeniser/2: PREDICATE

Usage: `tokeniser(VRML,Tokens)`

- *Description:* This predicate will perform the parsing of the VRML code. The result will be tokens that will be the source for producing the Prolog terms of the VRML code. This is done in the parser module. From these terms analysis, changing, and any thing that you want to do with VRML code from Prolog programming language. We perform the predicate with a catch call to be able to output error messages if encountered.
- *Call and exit should be compatible with:*
  `VRML` is a list of `atm`s. (basic_props:list/2)
  `Tokens` is a list of `term`s. (basic_props:list/2)
- *The following properties should hold at call time:*
  `VRML` is currently a term which is not a free variable. (term_typing:nonvar/1)
  `Tokens` is a free variable. (term_typing:var/1)

token_read/3: PREDICATE

Usage 1: `token_read(String,L,T)`

- *Description:* The predicate will return the token string with one argument the string. The string has some special properties according to the specification, but always starting with the string symbol.
- *Call and exit should be compatible with:*
  `String` is any term. (basic_props:term/1)
  `L` is a list. (basic_props:list/1)
  `T` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*
  `String` is a free variable. (term_typing:var/1)

Usage 2: `token_read(Comment,L,T)`

- *Description:* The predicate will return a token for a comment. The comment symbol, the bracket will start the comment and go for a complete line.
- *Call and exit should be compatible with:*

  `Comment` is any term. (basic_props:term/1)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)
- *The following properties should hold at call time:*

  `Comment` is a free variable. (term_typing:var/1)

Usage 3: `token_read(Id,L,T)`

- *Description:* The predicate will return a token for a word, an identifier. there is a check whether the id starts with a capital or a low cap letter. There are tough no distinction made in the token, but made a bit more easy to change if needed.
- *Call and exit should be compatible with:*

  `Id` is any term. (basic_props:term/1)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)

Usage 4: `token_read(Exp,L,T)`

- *Description:* The predicate will return the token for an exponentional number. The number are according to the definitions of VRML and due to the normal Prolog standard. We allow integer and float exponents but not exponential exponents.
- *Call and exit should be compatible with:*

  `Exp` is any term. (basic_props:term/1)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)

Usage 5: `token_read(Float,L,T)`

- *Description:* The predicate will return a token for a floating point value.
- *Call and exit should be compatible with:*

  `Float` is currently instantiated to a float. (term_typing:float/1)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)

Usage 6: `token_read(Hex,L,T)`

- *Description:* The returned token is of type hexadecimal number according to the specifications.E.g., 0xabcd3ef where the 0x is essential.
- *Call and exit should be compatible with:*

  `Hex` is any term. (basic_props:term/1)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)

Usage 7: `token_read(Int,L,T)`

- *Description:* The token returned is the integer token and will contain the integer value read.
- *Call and exit should be compatible with:*

  `Int` is an integer. (basic_props:int/1)

  `L` is a list. (basic_props:list/1)

  `T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Int` is a free variable. (term_typing:var/1)

Usage 8: `token_read(Symbol,L,T)`

– *Description:* The symbol returned with this token is one of the symbols allowed according to the specification. The only symbol is dot for the moment.

– *Call and exit should be compatible with:*

`Symbol` is any term. (basic_props:term/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

Usage 9: `token_read(ListOfType,L,T)`

– *Description:* The token for whitespace returns a list with the encountered whitespaces. This is a better way because a whitespace are selldom found alone. The whitespaces considered are: space, tab, new-line, return, and comma.

– *Call and exit should be compatible with:*

`ListOfType` is any term. (basic_props:term/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

Usage 10: `token_read(Paren,L,T)`

– *Description:* There are four tokens returned with this prefix: node_open, node_close, list_open, list_close.

– *Call and exit should be compatible with:*

`Paren` is an atom. (basic_props:atm/1)

`L` is a list. (basic_props:list/1)

`T` is a list. (basic_props:list/1)

– *The following properties should hold at call time:*

`Paren` is a free variable. (term_typing:var/1)

225 Pattern (regular expression) matching

Author(s): The CLIP Group.

This library provides facilities for matching strings and terms against *patterns*. There are some prolog flags

- There is a prolog flag to case insensitive match. Its name is case_insensitive. If its value is on, matching is case insenseitive, but if its value is off matching isn't case insensitive. By default, its value is off.
- There is a syntax facility to use matching more or less like a unification. You can type, " =~ "regexp" " as an argument of a predicate. Thus, that argument must match with regexp. For example:

```
pred ( =~ "ab*c", B) :- ...
```

  is equivalent to

```
pred (X,B) :- match_posix("ab*c",X,R), ...
```

  So, there are two prolog flags about this. One of this prolog flags is "format". Its values are shell, posix, list and pred, and sustitute in the example match_posix by match_shell, match_posix, match_struct and macth_pred respectivly. By default its value is posix. The other prolog flag is exact. Its values are on and off. If its value is off sustitute in the example R by []. If its value is on, R is a variable. By default, its value is on.

225.1 Usage and interface (regexp_doc)

- **Library usage:**
  `:- use_package(regexp).`
  or
  `:- module(...,...,[regexp]).`
- **New operators defined:**
  `=~/1` [200,fy].
- **Other modules used:**
  - *System library modules:*
    `regexp/regexp_code`.

225.2 Documentation on internals (regexp_doc)

match_shell/3: PREDICATE

Usage: `match_shell(Exp,IN,Rest)`

- *Description:* Matches `IN` against `Exp`. `Rest` is the longest remainder of the string after the match. For example, `match_shell("??*","foo.pl",Tail)` succeeds, instantiating `Tail` to `"o.pl"`.
- *The following properties should hold at call time:*
  `Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)
  `IN` is a string (a list of character codes). (basic_props:string/1)
  `Rest` is a string (a list of character codes). (basic_props:string/1)

match_shell/2: PREDICATE

Usage: `match_shell(Exp,IN)`

- *Description:* Matches completely `IN` (no tail can remain unmatched) against `Exp` similarly to `match_shell/3`.
- *The following properties should hold at call time:*

  `Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)

  `IN` is a string (a list of character codes). (basic_props:string/1)

match_posix/2: PREDICATE

Usage: `match_posix(Exp,IN)`

- *Description:* Matches completely `IN` (no tail can remain unmatched) against `Exp` similarly to `match_posix/3`.
- *The following properties should hold at call time:*

  `Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)

  `IN` is a string (a list of character codes). (basic_props:string/1)

match_posix/4: PREDICATE

Usage: `match_posix(Exp,In,Match,Rest)`

- *The following properties should hold at call time:*

  `Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)

  `In` is a string (a list of character codes). (basic_props:string/1)

  `Match` is a list of `string`s. (basic_props:list/2)

  `Rest` is a string (a list of character codes). (basic_props:string/1)

match_posix_rest/3: PREDICATE

Usage: `match_posix_rest(Exp,IN,Rest)`

- *Description:* Matches `IN` against `Exp`. `Tail` is the remainder of the string after the match. For example, `match_posix("ab*c","abbbbcdf",Tail)` succeeds, instantiating `Tail` to `"df"`.
- *The following properties should hold at call time:*

  `Exp` is a posix regular expression to match against. (regexp_code:posix_regexp/1)

  `IN` is a string (a list of character codes). (basic_props:string/1)

  `Rest` is a string (a list of character codes). (basic_props:string/1)

match_posix_matches/3: PREDICATE

Usage: `match_posix_matches(Exp,IN,Matches)`

- *Description:* Matches completely `IN` against `Exp`. `Exp` can contain *anchored expressions* of the form `\(regexp\)`. `Matches` will contain a list of the *anchored expression* which were matched on success. Note that since POSIX expressions are being read inside a string, backslashes will have to be doubled. For example,

```
?- match_posix_matches("\\(aa|bb\\)\\(bb|aa\\)", "bbaa", M).
M = ["bb","aa"] ? ;
no

?- match_posix_matches("\\(aa|bb\\)\\(bb|aa\\)", "aabb", M).
M = ["aa","bb"] ? ;
no
```

– *The following properties should hold at call time:*

`Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)

`IN` is a string (a list of character codes). (basic_props:string/1)

`Matches` is a list of `string`s. (basic_props:list/2)

match_struct/4: PREDICATE

Usage: `match_struct(Exp,IN,Rest,Tail)`

– *Description:* Matches `IN` against `Exp`. `Tail` is the remainder of the list of atoms `IN` after the match. For example, `match_struct([a,*(b),c],[a,b,b,b,c,d,e],Tail)` succeeds, instantiating `Tail` to `[d,e]`.

– *Call and exit should be compatible with:*

`Exp` is a struct regular expression to match against. (regexp_code:struct_regexp/1)

`IN` is a string (a list of character codes). (basic_props:string/1)

`Rest` is a string (a list of character codes). (basic_props:string/1)

match_pred/2: PREDICATE

Usage: `match_pred(Pred1,Pred2)`

– *Description:* Tests if two predicates `Pred1` and `Pred2` match using posix regular expressions.

replace_first/4: PREDICATE

Usage: `replace_first(IN,Old,New,Resul)`

– *Description:* Replace the first ocurrence of the `Old` by `New` in `IN` and copy the result in `Resul`.

– *The following properties should hold at call time:*

`IN` is a string (a list of character codes). (basic_props:string/1)

`Old` is a posix regular expression to match against. (regexp_code:posix_regexp/1)

`New` is a string (a list of character codes). (basic_props:string/1)

`Resul` is a string (a list of character codes). (basic_props:string/1)

replace_all/4: PREDICATE

Usage: `replace_all(IN,Old,New,Resul)`

– *Description:* Replace all ocurrences of the `Old` by `New` in `IN` and copy the result in `Resul`.

– *The following properties should hold at call time:*
`IN` is a string (a list of character codes). (basic_props:string/1)
`Old` is a posix regular expression to match against. (regexp_code:posix_regexp/1)
`New` is a string (a list of character codes). (basic_props:string/1)
`Resul` is a string (a list of character codes). (basic_props:string/1)

226 regexp_code (library)

226.1 Usage and interface (regexp_code)

- **Library usage:**
  `:- use_module(library(regexp_code)).`
- **Exports:**
  - *Predicates:*
    `match_shell/3`, `match_shell/2`, `match_posix/2`, `match_posix/4`, `match_posix_rest/3`, `match_posix_matches/3`, `match_struct/4`, `match_pred/2`, `replace_first/4`, `replace_all/4`.
  - *Regular Types:*
    `shell_regexp/1`, `posix_regexp/1`, `struct_regexp/1`.
  - *Multifiles:*
    `define_flag/3`.
- **Other modules used:**
  - *System library modules:*
    `lists`.

226.2 Documentation on exports (regexp_code)

match_shell/3: PREDICATE

Usage: `match_shell(Exp,IN,Rest)`

- *Description:* Matches `IN` against `Exp`. `Rest` is the longest remainder of the string after the match. For example, `match_shell("??*","foo.pl",Tail)` succeeds, instantiating `Tail` to `"o.pl"`.
- *The following properties should hold at call time:*
  `Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)
  `IN` is a string. (regexp_code:string/1)
  `Rest` is a string. (regexp_code:string/1)

match_shell/2: PREDICATE

Usage: `match_shell(Exp,IN)`

- *Description:* Matches completely `IN` (no tail can remain unmatched) against `Exp` similarly to `match_shell/3`.
- *The following properties should hold at call time:*
  `Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)
  `IN` is a string. (regexp_code:string/1)

match_posix/2: PREDICATE

Usage: `match_posix(Exp,IN)`

– *Description:* Matches completely `IN` (no tail can remain unmatched) against `Exp` similarly to `match_posix/3`.

– *The following properties should hold at call time:*

`Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)

`IN` is a string. (regexp_code:string/1)

match_posix/4: PREDICATE

Usage: `match_posix(Exp,In,Match,Rest)`

– *The following properties should hold at call time:*

`Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)

`In` is a string. (regexp_code:string/1)

`Match` is a list of `string`s. (basic_props:list/2)

`Rest` is a string. (regexp_code:string/1)

match_posix_rest/3: PREDICATE

Usage: `match_posix_rest(Exp,IN,Rest)`

– *Description:* Matches `IN` against `Exp`. `Tail` is the remainder of the string after the match. For example, `match_posix("ab*c","abbbbcdf",Tail)` succeeds, instantiating `Tail` to `"df"`.

– *The following properties should hold at call time:*

`Exp` is a posix regular expression to match against. (regexp_code:posix_regexp/1)

`IN` is a string. (regexp_code:string/1)

`Rest` is a string. (regexp_code:string/1)

match_posix_matches/3: PREDICATE

Usage: `match_posix_matches(Exp,IN,Matches)`

– *Description:* Matches completely `IN` against `Exp`. `Exp` can contain *anchored expressions* of the form `\(regexp\)`. `Matches` will contain a list of the *anchored expression* which were matched on success. Note that since POSIX expressions are being read inside a string, backslashes will have to be doubled. For example,

```
?- match_posix_matches("\\(aa|bb\\)\\(bb|aa\\)", "bbaa", M).
M = ["bb","aa"] ? ;
no

?- match_posix_matches("\\(aa|bb\\)\\(bb|aa\\)", "aabb", M).
M = ["aa","bb"] ? ;
no
```

– *The following properties should hold at call time:*

`Exp` is a shell regular expression to match against. (regexp_code:shell_regexp/1)

`IN` is a string. (regexp_code:string/1)

`Matches` is a list of `string`s. (basic_props:list/2)

match_struct/4: PREDICATE

Usage: `match_struct(Exp,IN,Rest,Tail)`

- *Description:* Matches `IN` against `Exp`. `Tail` is the remainder of the list of atoms `IN` after the match. For example, `match_struct([a,*(b),c],[a,b,b,b,c,d,e],Tail)` succeeds, instantiating `Tail` to `[d,e]`.
- *Call and exit should be compatible with:*

  `Exp` is a struct regular expression to match against. (regexp_code:struct_regexp/1)

  `IN` is a string. (regexp_code:string/1)

  `Rest` is a string. (regexp_code:string/1)

match_pred/2: PREDICATE

Usage: `match_pred(Pred1,Pred2)`

- *Description:* Tests if two predicates `Pred1` and `Pred2` match using posix regular expressions.

replace_first/4: PREDICATE

Usage: `replace_first(IN,Old,New,Resul)`

- *Description:* Replace the first ocurrence of the `Old` by `New` in `IN` and copy the result in `Resul`.
- *The following properties should hold at call time:*

  `IN` is a string. (regexp_code:string/1)

  `Old` is a posix regular expression to match against. (regexp_code:posix_regexp/1)

  `New` is a string. (regexp_code:string/1)

  `Resul` is a string. (regexp_code:string/1)

replace_all/4: PREDICATE

Usage: `replace_all(IN,Old,New,Resul)`

- *Description:* Replace all ocurrences of the `Old` by `New` in `IN` and copy the result in `Resul`.
- *The following properties should hold at call time:*

  `IN` is a string. (regexp_code:string/1)

  `Old` is a posix regular expression to match against. (regexp_code:posix_regexp/1)

  `New` is a string. (regexp_code:string/1)

  `Resul` is a string. (regexp_code:string/1)

shell_regexp/1: REGTYPE

Usage: `shell_regexp(P)`

- *Description:* `P` is a shell regular expression to match against.

posix_regexp/1: REGTYPE

Usage: `posix_regexp(P)`

- *Description:* `P` is a posix regular expression to match against.

struct_regexp/1: REGTYPE

Usage: `struct_regexp(P)`

– *Description:* `P` is a struct regular expression to match against.

226.3 Documentation on multifiles (`regexp_code`)

define_flag/3: PREDICATE

The predicate is *multifile*.

Usage: `define_flag(Flag,FlagValues,Default)`

– *The following properties hold upon exit:*

`Flag` is an atom. (basic_props:atm/1)

Define the valid flag values (basic_props:flag_values/1)

227 Automatic tester

Author(s): David Trallero Mena.

This module have been created to automate the test that a predicate should pass hopefully. With that intention we have to provide a set of test and its correct answers. The predicate `run_tester/10` will execute every test and compare it with its answer, generating two traces, one with detailed information, and another with the summary of executions of the tests.

227.1 Usage and interface (`tester`)

- **Library usage:**
  `:- use_module(library(tester)).`
- **Exports:**
  - *Predicates:*
    `run_tester/10`.
- **Other modules used:**
  - *System library modules:*
    `lists`, `write`, `io_alias_redirection`.

227.2 Documentation on exports (`tester`)

run_tester/10: PREDICATE

Meta-predicate with arguments: `run_tester(?,?,(pred 0),(pred 1),?,(pred 1),?,(pred 0),?,?)`.

Usage: `run_tester(LogFile,ResultFile,Begin,Test,TestList,Check,CheckList,End,GoodExamples,Slider`

- *Description:* run_tester is a predicate for automatizate testers. It get 2 file names as entry (`LogFile` and `ResultFile`) for saving the trace and the short result scheme respectevely. `Being` and `End` are called at the beginning and at the end of the test. `Test` is called which each element of `TestList` and after, `Check` is called with the corresponding element in `CheckList` for checking the results of `Test` predicate. `GoodExample` is ground(int) at the exit and tells the number of examples that passed the test correctly. `Slider` can take the values slider(no) or slider(Title) and slider will be shown everytime a new test is called
- *The following properties should hold at call time:*
  `LogFile` is a string (a list of character codes). (basic_props:string/1)
  `ResultFile` is a string (a list of character codes). (basic_props:string/1)
  `Begin` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  `Test` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  `TestList` is a list. (basic_props:list/1)
  `Check` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
  `CheckList` is a list. (basic_props:list/1)

`End` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`GoodExamples` is a free variable. (term_typing:var/1)

`Slider` is any term. (basic_props:term/1)

227.3 Other information (`tester`)

Two simple examples of the use of the run_tester are provided.

227.3.1 Understanding run_test predicate

```
:- module( tester_test2 , _ , _ ).


:- use_module('..'(tester)).
%:- use_module( library(tester) ).
:- use_module(library(lists)).
:- use_module(library(write)).

init_func :-
        write( 'Starting the test\n' ).

tester_func( (X,X,_) ) :-
        write( 'The argument is correct ' ),
        write( X ) , nl.

checker_func( (_,X,X) ) :-
        write( 'check is fine\n\n' ).


end_func :-
        write( 'Test ended\n' ).


main :-
        L = [ (1,1,1),   % CORRECT
              (2,2,1),   % Test CORRECT , CHECK FALSE
              (1,2,2)    % Test FALSE
            ],

         run_tester(
                     'test.log',
                     'result.log',
                     init_func ,
                     tester_func ,
                     L,
                     checker_func,
                     L,
                     end_func,
                     Res,
                     slider( 'Tester2: ' )
```

```
                ),

        length( L , LL ),
        Op is (Res / LL) * 100,
        message( note , [ 'Analysis result: ' , Op , '%' ] ).
```

227.3.2 More complex example

In this example we just want to test if the output of Ciaopp is readable by CIAO.
Tester function succeds if it is able to write the output file.
Checker function succeds if it is able to load the written file.

```
:- module( tester_test1 , _ , [] ).

%:- use_module( library(tester) , [run_tester/10] ).
:- use_module('..'(tester), [run_tester/10]).


:- use_module(library(ciaopp)).
:- use_module(library(compiler)).

:- use_module(library(filenames)).

:- use_module(library(write)).

:- use_module(library(lists)).

init_func.


test_files( '/home/dtm/Ciaopp/Benchmarks/ciaopp/modes/' ).

tester_func( FileArg ) :-
        test_files( Path ),
        atom_concat( Path , FileArg , File0 ),

        message( note ,
         [ '++++++++++++++++++++++++++++++++++++++++++++++\n' ] ),
        (unload( File0 )->true;true),
        module( File0 ),

        atom_concat( TFile , '.pl', File0 ),
        atom_concat( TFile , '_test.pl' , TestFile ),

        output( TestFile ).


get_module( Path , Module ) :-
        no_path_file_name( Path , File ),
        (atom_concat( Module , '.pl' , File )
        -> true ; Module = File ).
```

```
checker_func( FileArg ) :-
        get_module( FileArg , Module ),
        (unload( Module )->true;true),

        atom_concat(RawFile, '.pl'     , FileArg ),
        atom_concat(RawFile, '_test.pl' , OptFile ),

        test_files( Path ),
        atom_concat( Path , OptFile, OptFilePath ),

        message( note , [ 'Cargando ' , OptFilePath ] ),
        use_module( OptFilePath ).

end_func.




main :-
        L = [
                'aiakl.pl',
                'query.pl',
                'mmatrix.pl',
                'ann.pl',
                'bid.pl',
                'rdtok.pl',
                'myread.pl',
                'boyer.pl',
                'read.pl',
                'occur.pl',
                'serialize.pl',
                'browse.pl',
                'peephole.pl',
                'tak.pl',
                'deriv.pl',
                'progeom.pl',
                'warplan.pl',
                'fib.pl',
                'qplan.pl',
                'witt.pl',
                'grammar.pl',
                'zebra.pl',
                'qsortapp.pl',
                'hanoiapp.pl'
              ],

        run_tester(
                        'test.log',
                        'result.log',
```

```
                init_func ,
                tester_func ,
                L,
                checker_func,
                L,
                end_func,
                Res,
                slider( 'Tester1: ' )
            ),
length( L , LL ),
Op is (Res / LL) * 100,
message( note , [ 'Analysis result: ' , Op , '%' ] ).
```

228 Measuring features from predicates (time cost or memory used)

Author(s): David Trallero Mena.

This library has been done for measuring or compare execution features (currently only time) of predicates. This module relys on gnuplot, an auxiliary module which use the tool `gnuplot`, for representing results graphically

228.1 Usage and interface (`time_analyzer`)

- **Library usage:**
  `:- use_module(library(time_analyzer)).`
- **Exports:**
  - *Predicates:*
    `performance/3`, `benchmark/6`, `compare_benchmark/7`, `generate_benchmark_list/7`, `benchmark2/6`, `compare_benchmark2/7`, `generate_benchmark_list2/7`, `sub_times/3`, `div_times/2`, `cost/3`.
- **Other modules used:**
  - *System library modules:*
    `gnuplot/gnuplot`, `prolog_sys`, `lists`, `write`, `hiordlib`.

228.2 Documentation on exports (`time_analyzer`)

performance/3: PREDICATE

Meta-predicate with arguments: `performance(goal,?,?)`.

Usage: `performance(P,M,Times)`

- *Description:* performance accepts a goal, `P`, as a first argument. The aim of this predicate is to call `P` several times and meassure some feature (in this version, only time, that is reason because no extra parameter has been added). `M` defines how many times `P` should be called. Usually, calling the predicate in some succesion (10,100,1000) and dividing by the number of times it is executed we can obtain the "execution time" of the predicate (if we are measuring time).

  The result of executions are returned in the list `Times`

  The diferent modes are:

  - graph(Start , End , Increment). It defines arithmetic succesion starting in Start and ending in End, by increment of Increment. So `P` is called Start times on the first time, Start+Increment on the second, etc.
  - graph The same as graph/3 but with default options
  - graph_exp(Start , End , Exp). It defines geometric succesion. Start is multiplied by Exp till it gets End. So `P` is called Start times on the first time, Start*Exp on the second, etc.
  - graph_exp The same as graph_exp/3 but with default options
- *The following properties should hold at call time:*

  `P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`M` is any term. (basic_props:term/1)

`Times` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`M` is any term. (basic_props:term/1)

`Times` is a list of `num`s. (basic_props:list/2)

benchmark/6: PREDICATE

Usage: `benchmark(P,BenchList,NumTimes,Method,Reserved,OutList)`

– *Description:* The predicate `P`, which accepts ONE argument, is called with the first member of each pair of the `BenchList` list `NumTimes`. The entry list have pairs because the second member of the pair express the meaning of the first one in the X-Axis. For example, if we are doing a benchmark of qsort function, the first member will be a list for being ordered and the second one will be the length of the unordered list. The output is a list of (X,Y) points where Y means the time needed for its entry of "cost" X. `OutList` can be used as TimeList in predicate generate_plot. `Reserved` is reserved for future implementations (it will take the value of runtime, memory_used...)

– *The following properties should hold at call time:*

`P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`BenchList` is a list of `pair`s. (basic_props:list/2)

`NumTimes` is an integer. (basic_props:int/1)

time_analyzer:average_mode(Method) (time_analyzer:average_mode/1)

`Reserved` is any term. (basic_props:term/1)

`OutList` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`BenchList` is a list of `pair`s. (basic_props:list/2)

`NumTimes` is an integer. (basic_props:int/1)

time_analyzer:average_mode(Method) (time_analyzer:average_mode/1)

`Reserved` is any term. (basic_props:term/1)

`OutList` is a list of `pair`s. (basic_props:list/2)

compare_benchmark/7: PREDICATE

Usage: `compare_benchmark(ListPred,BenchList,Method,NumTimes,BaseName,Reserved,GeneralOptions)`

– *Description:* It is the generalization of execute predicate `benchmark/6` with several predicates. `benchmark/6` predicate is called with each predicate in `ListPred`, and `BaseName` is used for the temporaries basename file. `GeneralOptions` are aplied to the plot

– *The following properties should hold at call time:*

`ListPred` is a list of `callable`s. (basic_props:list/2)

`BenchList` is a list. (basic_props:list/1)

time_analyzer:average_mode(Method) (time_analyzer:average_mode/1)
`NumTimes` is an integer. (basic_props:int/1)
`BaseName` is currently instantiated to an atom. (term_typing:atom/1)
`Reserved` is any term. (basic_props:term/1)
`GeneralOptions` is a list. (basic_props:list/1)

generate_benchmark_list/7: PREDICATE

No further documentation available for this predicate.

benchmark2/6: PREDICATE

Usage: `benchmark2(P,BenchList,Method,NumTimes,What,OutList)`

– *Description:* The predicate `P`, which accepts TWO arguments, is called `NumTimes` with the first member of each pair of the `BenchList` list and a free variable as the second. The time of execution (in the future, the desired featured for be measured) is expected to be the second argument, that is because it is a variable. The entry list, `BenchList` have pairs because the second member of the pair express the cost of the first (in X-Axis). For example, if we are doing a benchmark of qsort function, the first member will be a list for being ordered and the second one will represent the lenght of the unordered list. The output is a list of (X,Y) points where Y express the time needed for they entry of `"cost"` X. `OutList` can be used as TimeList in predicate generate_plot. `What` is reserved for future use

– *The following properties should hold at call time:*

`P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
`BenchList` is a list of `pair`s. (basic_props:list/2)
time_analyzer:average_mode(Method) (time_analyzer:average_mode/1)
`NumTimes` is an integer. (basic_props:int/1)
`What` is currently instantiated to an atom. (term_typing:atom/1)
`OutList` is a free variable. (term_typing:var/1)

– *The following properties should hold upon exit:*

`P` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)
`BenchList` is a list of `pair`s. (basic_props:list/2)
time_analyzer:average_mode(Method) (time_analyzer:average_mode/1)
`NumTimes` is an integer. (basic_props:int/1)
`What` is currently instantiated to an atom. (term_typing:atom/1)
`OutList` is a list of `pair`s. (basic_props:list/2)

compare_benchmark2/7: PREDICATE

Usage: `compare_benchmark2(ListPred,BenchList,Method,NumTimes,BaseName,Reserved,GeneralOptions)`

– *Description:* It is the generalization of execute predicate `benchmark2/6` with several predicates. `benchmark2/6` is called with each predicate in `ListPred` and `BaseName` is used for the temporaries basename file. `GeneralOptions` are applied to the plot ('default' can be used for default General options)

– *The following properties should hold at call time:*

`ListPred` is a list of `callable`s. (basic_props:list/2)
`BenchList` is a list. (basic_props:list/1)
time_analyzer:average_mode(Method) (time_analyzer:average_mode/1)
`NumTimes` is an integer. (basic_props:int/1)
`BaseName` is currently instantiated to an atom. (term_typing:atom/1)
`Reserved` is currently instantiated to an atom. (term_typing:atom/1)
`GeneralOptions` is a list. (basic_props:list/1)

– *The following properties should hold upon exit:*

`ListPred` is a list of `callable`s. (basic_props:list/2)
`BenchList` is a list. (basic_props:list/1)
time_analyzer:average_mode(Method) (time_analyzer:average_mode/1)
`NumTimes` is an integer. (basic_props:int/1)
`BaseName` is currently instantiated to an atom. (term_typing:atom/1)
`Reserved` is currently instantiated to an atom. (term_typing:atom/1)
`GeneralOptions` is a list. (basic_props:list/1)

generate_benchmark_list2/7: PREDICATE

No further documentation available for this predicate.

sub_times/3: PREDICATE

Usage: `sub_times(A,B,C)`

– *Description:* C is the result of doing A - B, where A, B, C are a list of pairs as (Time,_)

– *Call and exit should be compatible with:*

`A` is a list of `pair`s. (basic_props:list/2)
`B` is a list of `pair`s. (basic_props:list/2)
`C` is a list of `pair`s. (basic_props:list/2)

div_times/2: PREDICATE

Usage: `div_times(A,B)`

– *Description:* `A` is a list of pairs (P1,P2). `B` is a list of pairs with the form (P1,P2/P1) for each (P1,P2) that belongs to `A`

– *Call and exit should be compatible with:*

`A` is a list of `pair`s. (basic_props:list/2)
`B` is a list of `pair`s. (basic_props:list/2)

cost/3: PREDICATE

Meta-predicate with arguments: `cost(goal,?,?)`.

Usage: `cost(A,T,What)`

– *Description:* This pred is thought for measuring constant complexity predicates. `T` is the expected measured feature. `What` is reserved for future implementations, just put 'runtime'
– *Call and exit should be compatible with:*

`A` is a term which represents a goal, i.e., an atom or a structure. (basic_props:callable/1)

`T` is an integer. (basic_props:int/1)

`What` is any term. (basic_props:term/1)

generate_plot/3: (UNDOC_REEXPORT)
Imported from `gnuplot` (see the corresponding documentation for details).

generate_plot/2: (UNDOC_REEXPORT)
Imported from `gnuplot` (see the corresponding documentation for details).

set_general_options/1: (UNDOC_REEXPORT)
Imported from `gnuplot` (see the corresponding documentation for details).

get_general_options/1: (UNDOC_REEXPORT)
Imported from `gnuplot` (see the corresponding documentation for details).

229 XDR handle library

Author(s): José Manuel Gómez Pérez.

This library offers facilities to enable users to setup preferences on the values an eventual XML document may take. XML documents are specified by XDR documents (eXternal Data Representation standard), in a way conceptually similar to that of objects and classes in object oriented programming. These facilities allow to take as input an XDR Schema defining the class of documents of interest, and establish a dialogue with the user via an HTML form that allows the user to setup preferences to select sub-classes of documents (those which satisfy the preferences). The preferences are the output of the process and may be in the form of XPath expressions, for example, as can be seen in the example attached in the "examples" directory.

229.1 Usage and interface (xdr_handle)

- **Library usage:**
  `:- use_module(library(xdr_handle)).`
- **Exports:**
  - *Predicates:*
    `xdr_tree/3`, `xdr_tree/1`, `xdr2html/4`, `xdr2html/2`, `unfold_tree/2`, `unfold_tree_dic/3`, `xdr_xpath/2`.
  - *Regular Types:*
    `xdr_node/1`.
- **Other modules used:**
  - *System library modules:*
    `pillow/http`, `pillow/html`, `pillow/pillow_types`, `xdr_handle/xdr_types`, `aggregates`, `lists`, `terms`.

229.2 Documentation on exports (xdr_handle)

xdr_tree/3: PREDICATE

Usage: `xdr_tree(XDR_url,XDR_tree,XDR_id)`

- *Description:* Parses an XDR (External Data Representation Standard) located at an url `XDR_url` into a tree structured Prolog term `XDR_tree`. It also returns an identifier of the XDR_tree `XDR_id` corresponding to the sequence of nodes in the tree (this is intended to be a hook to use in CGI applications).
- *The following properties should hold at call time:*

`XDR_url` is currently a term which is not a free variable.	(term_typing:nonvar/1)
`XDR_tree` is a free variable.	(term_typing:var/1)
`XDR_id` is a free variable.	(term_typing:var/1)
`XDR_url` specifies a URL.	(pillow_types:url_term/1)
`XDR_tree` specifies an XDR document.	(xdr_types:xdr/1)
`XDR_id` is an integer.	(basic_props:int/1)

xdr_tree/1: PREDICATE

Usage: `xdr_tree(XDR_tree)`

- *Description:* Checks the correctness of an XDR tree `XDR_tree`.
- *The following properties should hold at call time:*

  `XDR_tree` specifies an XDR document. (xdr_types:xdr/1)

xdr_node/1: REGTYPE

Usage: `xdr_node(XDR_node)`

- *Description:* `XDR_node` is a XDR tree node.

xdr2html/4: PREDICATE

Usage: `xdr2html(XDRTree,HTMLOutput,UnfoldedTree,Dic)`

- *Description:* Receives an XDR tree `XDRTree` and produces the corresponding HTML code `HTMLOutput`, an equivalente unfolded plain tree `UnfoldedTree` and a control dictionary `Dic` to hold a reference the evenutal fom objects.
- *The following properties should hold at call time:*

  `XDRTree` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `HTMLOutput` is a free variable. (term_typing:var/1)

  `UnfoldedTree` is a free variable. (term_typing:var/1)

  `Dic` is a free variable. (term_typing:var/1)

  `XDRTree` specifies an XDR document. (xdr_types:xdr/1)

  `HTMLOutput` is a term representing HTML code. (pillow_types:html_term/1)

  `UnfoldedTree` specifies an XDR document. (xdr_types:xdr/1)

  `Dic` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (pillow_types:form_dict/1)

xdr2html/2: PREDICATE

Usage: `xdr2html(XDRTree,HTMLOutput)`

- *Description:* Receives an XDR tree `XDRTree` and produces the corresponding HTML code `HTMLOutput`. This html code is intended to be part of a form used as a means by which an eventual user can give value to an instance of the XDR, i.e. an XML element.
- *The following properties should hold at call time:*

  `XDRTree` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `HTMLOutput` is a free variable. (term_typing:var/1)

  `XDRTree` specifies an XDR document. (xdr_types:xdr/1)

  `HTMLOutput` is a term representing HTML code. (pillow_types:html_term/1)

unfold_tree/2: PREDICATE

Usage: `unfold_tree(XDRTree,UFT)`

– *Description:* Obtains an unfolded XDR tree `UFT` from a standard XDR tree `XDRTree`, i.e. an XDR tree where all references to XDR elements have been substituted with the elements themselves. Especially useful for eventual generation of equivalent XPATH expressions, (see example).
– *The following properties should hold at call time:*

`XDRTree` is currently a term which is not a free variable. (term_typing:nonvar/1)
`UFT` is a free variable. (term_typing:var/1)
`XDRTree` specifies an XDR document. (xdr_types:xdr/1)
`UFT` specifies an XDR document. (xdr_types:xdr/1)

unfold_tree_dic/3: PREDICATE

Usage: `unfold_tree_dic(XDRTree,UFT,Dic)`

– *Description:* Obtains an unfolded XDR tree `UFT` and a form dictionary `Dic` from a standard XDR tree `XDRTree`. Especially useful for HTML form data exchange (see example).
– *The following properties should hold at call time:*

`XDRTree` is currently a term which is not a free variable. (term_typing:nonvar/1)
`UFT` is a free variable. (term_typing:var/1)
`Dic` is a free variable. (term_typing:var/1)
`XDRTree` specifies an XDR document. (xdr_types:xdr/1)
`UFT` specifies an XDR document. (xdr_types:xdr/1)
`Dic` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (pillow_types:form_dict/1)

xdr_xpath/2: PREDICATE

Usage: `xdr_xpath(XDRTree,XPath)`

– *Description:* Produces an XPATH expression `XPath` from an XDR tree `XDRTree`. If the given XDR tree has no definite value the xpath expression produced will be empty
– *The following properties should hold at call time:*

`XDRTree` is currently a term which is not a free variable. (term_typing:nonvar/1)
`XPath` is a free variable. (term_typing:var/1)
`XDRTree` specifies an XDR document. (xdr_types:xdr/1)
`XPath` is an atom. (basic_props:atm/1)

230 XML query library

Author(s): José Manuel Gómez Pérez.

This package provides a language suitable for querying XML documents from a Prolog program. Constraint programming expresions can be included in order to prune search as soon as possible, i.e. upon constraint unsatisfability, improving efficiency. Also, facilities are offered to improve search speed by transforming XML documents into Prolog programs, hence reducing search to just running the program and taking advantage of Prolog's indexing capabilities.

Queries in an XML document have a recursive tree structructure that permits to detail the search on the XML element sought, its attributes, and its children. As a suffix, a constraint programming expression can be added. Queries return value for the free variables included (in case of success), and checks whether the XML document structure matches that depicted by the query itself.

The operators introduced are described below:

- `@` Delimits a subquery on an elment's attribute, such as `product@val(product_name, "car")`, the first argument being the attribute name and the second its value. Any of them can be free variables, being possible to write queries like `product@val(Name, "car")`, intended to find the 'Name' of attributes of element product whose value is the string `"car"`.
- *::* The right-hand side of the subexpression delimited by this operator is a query on the children elements of the element described on its left-hand side.
- *with* Declares the constraints the items sought must satisfy.

Some examples of this query language (more can be found in the examples directory):

- Example A:

```
product@val(product_name,"car")::(quantity(X),
                                  'time-left'(Y),
                                  negotiation::preference::price(Z))
        with X * Z .>. Y
```

- Example B:

```
nitf::head::docdata::'doc-id'@val('id-string',"020918050")::(Y),
                     body::'body.head'::abstract::p(X)
```

230.1 Usage and interface (`xml_path_doc`)

- **Library usage:**

  `:- use_package(xml_path).`

  or

  `:- module(...,...,[xml_path]).`
- **Exports:**
  - *Predicates:*

    `xml_search/3`, `xml_parse/3`, `xml_parse_match/3`, `xml_search_match/3`, `xml_index_query/3`, `xml_index_to_file/2`, `xml_index/1`, `xml_query/3`.
- **Other modules used:**
  - *System library modules:*

    `xml_path/xml_path_types`.

230.2 Documentation on exports (xml_path_doc)

xml_search/3: PREDICATE

Usage: `xml_search(Query,Source,Doc)`

- *Description:* Checks a high level query `Query` against an XML document `Source`. If the query is successful it retuns in `Doc` the whole xml element(s) of the document that matched it.
- *The following properties should hold at call time:*

  `Query` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Source` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Doc` is a free variable. (term_typing:var/1)

  `Query` is a primitive XML query. (xml_path_types:canonic_xml_query/1)

  `Source` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

  `Doc` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_parse/3: PREDICATE

Usage: `xml_parse(Query,Source,Doc)`

- *Description:* Checks a high level query `Query` against an XML document `Source`. If the query is successful it retuns in `Doc` the whole xml element(s) of the document that matched it. On the contrary as `xml_search/3`, the query can start at any level of the XML document, not necessarily at the root node.
- *The following properties should hold at call time:*

  `Query` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Source` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Doc` is a free variable. (term_typing:var/1)

  `Query` is a primitive XML query. (xml_path_types:canonic_xml_query/1)

  `Source` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

  `Doc` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_parse_match/3: PREDICATE

Usage: `xml_parse_match(Query,Source,Match)`

- *Description:* Checks a high level query `Query` against an XML document `Source`. If the query is successful it retuns in `Doc` the exact subtree of the xml document that matched it. On the contrary as `'$xml_search_match/3`, the query can start at any level of the XML document, not necessarily at the root node.
- *The following properties should hold at call time:*

  `Query` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Source` is currently a term which is not a free variable. (term_typing:nonvar/1)

  `Match` is a free variable. (term_typing:var/1)

  `Query` is a primitive XML query. (xml_path_types:canonic_xml_query/1)

`Source` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

`Match` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_search_match/3: PREDICATE

Usage: `xml_search_match(BasicQuery,SourceDoc,Match)`

- *Description:* Checks query `Query` against an XML document `Source`. If the query is successful it retuns in `Doc` the exact subtree of the xml document that matched it.
- *The following properties should hold at call time:*

`BasicQuery` is currently a term which is not a free variable. (term_typing:nonvar/1)

`SourceDoc` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Match` is a free variable. (term_typing:var/1)

`BasicQuery` is a primitive XML query. (xml_path_types:canonic_xml_query/1)

`SourceDoc` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

`Match` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_index_query/3: PREDICATE

Usage: `xml_index_query(Query,Id,Match)`

- *Description:* Matches a high level query `Query` against an XML document previously transformed into a Prolog program. `Id` identifies the resulting document `Match`, which is the exact match of the query against the XML document.
- *The following properties should hold at call time:*

`Query` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Id` is a free variable. (term_typing:var/1)

`Match` is a free variable. (term_typing:var/1)

`Query` is a primitive XML query. (xml_path_types:canonic_xml_query/1)

`Id` is an atom. (basic_props:atm/1)

`Match` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_index_to_file/2: PREDICATE

Usage: `xml_index_to_file(SourceDoc,File)`

- *Description:* Transforms the XML document `SourceDoc` in a Prolog program which is output to file `File`.
- *The following properties should hold at call time:*

`SourceDoc` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

`File` is an atom. (basic_props:atm/1)

xml_index/1: PREDICATE

Usage: `xml_index(SourceDoc)`

– *Description:* Transforms the XML document `SourceDoc` in a Prolog program, generating the associated clauses, which are stored dynamically into the current process memory space.

– *The following properties should hold at call time:*

`SourceDoc` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

xml_query/3: PREDICATE

Usage: `xml_query(Query,Doc,Match)`

– *Description:* Checks that XML document `Doc` is compliant with respect to the query `Query` expressed in the low level query language. The exact mapping of the query over the document is returned in `Match`

– *The following properties should hold at call time:*

`Query` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Doc` is currently a term which is not a free variable. (term_typing:nonvar/1)

`Match` is a free variable. (term_typing:var/1)

`Query` is a primitive XML query. (xml_path_types:canonic_xml_query/1)

`Doc` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

`Match` is either a XML attribute, a XML element or a line break. (xml_path_types:canonic_xml_item/1)

230.3 Documentation on internals (`xml_path_doc`)

canonic_xml_term/1: REGTYPE

Usage: `canonic_xml_term(XMLTerm)`

– *Description:* `XMLTerm` is a term representing XML code in canonical form.

canonic_xml_item/1: REGTYPE

Usage: `canonic_xml_item(XMLItem)`

– *Description:* `XMLItem` is either a XML attribute, a XML element or a line break.

tag_attrib/1: REGTYPE

Usage: `tag_attrib(Att)`

– *Description:* `Att` is a XML attribute.

canonic_xml_query/1: REGTYPE

Usage: `canonic_xml_query(Query)`

– *Description:* `Query` is a primitive XML query.

canonic_xml_subquery/1: REGTYPE

Usage: `canonic_xml_subquery(SQuery)`

– *Description:* `SQuery` defines a XML subquery.

PART XII - Appendices

Author(s): The CLIP Group.

These appendices describe the installation of the Ciao environment on different systems and some other issues such as reporting bugs, signing up on the Ciao user's mailing list, downloading new versions, limitations, etc.

231 Installing Ciao from the source distribution

Author(s): Manuel Carro, Daniel Cabeza, Manuel Hermenegildo.

This describes the installation procedure for the Ciao system, including libraries and manuals, from a *source* distribution. This applies primarily to Unix-type systems (Linux, Mac OS X, Solaris, SunOS, etc.). However, the sources can also be compiled if so desired on Windows systems – see Section 231.6 [Installation and compilation under Windows], page 1107 for details.

If you find any problems during installation, please refer to Section 231.8 [Troubleshooting (nasty messages and nifty workarounds)], page 1108. See also Section 233.3 [Downloading new versions], page 1115 and Section 233.4 [Reporting bugs], page 1116.

231.1 Un*x installation summary

Note: It is recommended that you read the full installation instructions (specially if the installation will be shared by different architectures). However, in many cases it suffices to follow this summary:

1. Uncompress and unpackage (using `gunzip` and `tar -xpf`) the distribution. This will put everything in a new directory whose name reflects the Ciao version.
2. Type `./ciaosetup configure`. This will autodetect and configure the system for your specific platform. If you need more control about what configure does, just add the option `--help` to see more arguments.

   The option `--menu` show a menu with configurable options. You must follow the instructions that appears on it.

   Note that the GNU implementation of the make Un*x command is used internally in some specific parts. It is available in many systems (including all Linux systems and Mac OS X) simply as `make`. If this or next steps stop right away with `make` error messages it is probably an older version and you need to install `gmake`.
3. Type `./ciaosetup build`. This will build executables and compile libraries.
4. Type `./ciaosetup install`. This will install everything in the specified directories.
5. The system will do the right modifications in your startup scripts. This will make the documentation accessible, set the correct mode when opening Ciao source files in `emacs`, etc. The modified files are tagged with the names \'DOTBASHRC\', \'DOTCSHRC\' and \'DOTEMACS\', which are the startup files for bash, csh and emacs respectively.

   The following modifications are done automatically in your startup scripts, and you don't need to do it manually. This will make the documentation accessible, set the correct mode when opening Ciao source files in `emacs`, etc. Note that `<v>libroot</v>` must be replaced with the appropriate value:

   - For users a *csh-compatible shell* (`csh`, `tcsh`, ...), add to `~/.cshrc`:

     ```
     if ( -e <v>libroot</v>/ciao/DOTcshrc ) then
        source <v>libroot</v>/ciao/DOTcshrc
     endif
     ```

     Note: while this is recognized by the terminal shell, and therefore by the text-mode Emacs which comes with Mac OS X, the Aqua native Emacs 21 does not recognize that initialization. It is thus necessary, at this moment, to set manually the Ciao shell (ciaosh) and Ciao library location by hand. This can be done from the Ciao menu within Emacs after a Ciao file has been loaded. We believe that the reason is that Mac OS X does not actually consult the per-user initialization files on startup. It should also be possible to put the right initializations in the .emacs file using the `setenv` function of Emacs-lisp, as in

```
(setenv "CIAOLIB" "<v>libroot</v>/ciao")
```

The same can be done for the rest of the variables initialized in `<v>libroot</v>/ciao/DOTcshrc`

- For users of an *sh-compatible shell* (`sh`, `bash`, ...), the installer will add to `~/.bashrc` the next lines:

```
if [ -f <v>libroot</v>/ciao/DOTprofile ]; then
   . <v>libroot</v>/ciao/DOTprofile
fi
```

This will set up things so that the Ciao executables are found and you can access the Ciao system manuals using the `info` command. Note that, depending on your shell, *you may have to log out and back in* for the changes to take effect.

- Also, if you use `emacs` (highly recommended) the install will add the next line to your `~/.emacs` file:

```
(load-file "<v>libroot</v>/ciao/ciao-mode-init.el")
(if (file-exists-p "<v>libroot</v>/ciao/ciao-mode-init.el")
  (load-file "<v>libroot</v>/ciao/ciao-mode-init.el")
)
```

If you are installing Ciao globally in a multi-user machine, make sure that you instruct all users to do the same. If you are the system administrator, the previous steps can be done once and for all, and globally for all users by including the lines above in the central startup scripts (e.g., in Linux `/etc/bashrc`, `/etc/csh.login`, `/etc/csh.cshrc`, `/etc/skel`, `/usr/share/emacs/.../lisp/site-init.pl`, etc.).

6. Finally, if the (freely available) `emacs` editor/environment is not installed in your system, we *highly recommend* that you also install it at this point (see Section 231.2 [Un*x full installation instructions], page 1102 for instructions). While it is easy to use Ciao with any editor of your choice, the Ciao distribution includes a very powerful *application development environment* which is based on `emacs` and which enables, e.g., source-level debugging, syntax coloring, context-sensitive on-line help, etc.
7. You may want now want to check your installation (see Section 231.3 [Checking for correct installation on Un*x], page 1105) and read the documentation, which is stored in `DOCDIR` (copied from `CIAOSRC/doc/reference`) and can be easily accessed as explained in that same section. There are special "getting started" sections at the beginning of the manual.
8. If you have any problems you may want to check the rest of the instructions. The system can be *uninstalled* by typing `./ciaosetup uninstall`.

231.2 Un*x full installation instructions

1. **Uncompress and unpackage:** (using `gunzip` and `tar -xpf`) the distribution in a suitable directory. This will create a new directory called `ciao-X.Y`, where `X.Y` is the version number of the distribution. The `-p` option in the `tar` command ensures that the relative dates of the files in the package are preserved, which is needed for correct operation of the Makefiles.
2. **Select installation options:** Run the `./ciaosetup configure` command and answer the questions that appears in the menu. The meaning of some important options in the menu is as follows:

   %

   - `CIAOSRC`: directory where the sources are % stored.
   - `BINROOT`: directory where the Ciao executables will go. For example, if `BINROOT=/usr/local/bin`, then the Ciao compiler (`ciaoc`) will be stored at

`/usr/local/bin/ciaoc`. Actually, it will be a link to `ciaoc-`*VersionNumber*. This applies also to other executables below and is done so that several versions of Ciao can coexist on the same machine. Note that the *version installed latest* will be the one started by default when typing `ciao`, `ciaoc`, etc.

- `LIBROOT`: directory where the run-time libraries will be installed. The Ciao installation procedure will create a new subdirectory `ciao` below `LIBROOT` and a subdirectory below this one for each Ciao version installed. For example, if `LIBROOT=/usr/local/lib` and you have Ciao version `x.y`, then the libraries will be installed under `/usr/local/lib/ciao/ciao-x.y`. This allows you to install site-specific programs under `/usr/local/lib/ciao` and they will not be overwritten if a new version of Ciao is installed. It also again allows having several Ciao versions installed simultaneously.
- `DOCDIR`: directory where the manuals will be installed. It is often convenient if this directory is accessible via WWW (`DOCDIR=/home/httpd/html/ciao`, or something like that).

For network-based installations, it is of *utmost importance* that the configured paths be reachable in all the networked machines. Different machines with different architectures can share the same physical source directory during installation, since compilations for different architectures take place in dedicated subdirectories. Also, different machines/architectures can share the same `LIBROOT` directory (`LIBROOT` is configured in the menu). This saves space since the architecture-independent libraries will be shared. See Section 231.5 [Multi-architecture support], page 1106 below.

3. **Compile Ciao:** At the ciao top level directory type `./ciaosetup build`.

   This will:

   - Build an engine in `$(CIAOSRC)/bin/$(CIAOARCH)`, where `$(CIAOARCH)` depends on the architecture. The engine is the actual interpreter of the low level code into which Ciao programs are compiled.
   - Build a new Ciao standalone compiler (`ciaoc`), with the default paths set for your local configuration (nonetheless, these can be overridden by environment variables, as described below).
   - Precompile all the libraries under `$(CIAOSRC)/lib` and `$(CIAOSRC)/library` using this compiler.
   - Compile a toplevel Ciao shell and a shell for Ciao scripts, under the `$(CIAOSRC)/shell` directory.
   - Compile some small, auxiliary applications (contained in the `etc` directory, and documented in the part of the manual on 'Miscellaneous Standalone Utilities').

   This step can be repeated successively for several architectures in the same source directory. Only the engine and some small parts of the libraries (those written in C) differ from one architecture to the other. Standard Ciao code compiles into bytecode object files (`.po`) and/or executables which are portable among machines of different architecture, provided there is an executable engine accessible in every such machine. See more details below under Section 231.5 [Multiarchitecture support], page 1106.
4. **Check compilation:** If the above steps have been satisfactorily finished, the compiler has compiled itself and all the distribution modules, and very probably everything is fine.
5. **Install Ciao:** To install Ciao in the directories selected in the configuration script during step 2 above, type `./ciaosetup install`. This will:

   - Install the executables of the Ciao program development tools (i.e., the general driver/top-level `ciao`, the standalone compiler `ciaoc`, the script interpreter `ciao-shell`, miscellaneous utilities, etc.) in `BINROOT` (see below). In order to use these tools, the `PATH` environment variable of users needs to contain the path `BINROOT`.

- Install the Ciao libraries under `LIBROOT/ciao` (these will be automatically found).
- Install under `DOCDIR` the Ciao manuals in several formats (such as GNU `info`, `html`, `postscript`, etc.), depending on the distribution. In order for these manuals to be found when typing `M-x info` within `emacs`, or by the standalone `info` and `man` commands, the `MANPATH` and `INFOPATH` environment variables of users both need to contain the path `DOCDIR`.
- Install under `LIBROOT/ciao` the Ciao GNU `emacs` interface (`ciao.el`, which provides an interactive interface to the Ciao program development tools, as well as some other auxiliary files) and a file `ciao-mode-init` containing the `emacs` initialization commands which are needed in order to use the Ciao `emacs` interface.

6. **Set up user environments:** In order to automate the process of setting the variables above, the installation process leaves the files `LIBROOT/ciao/DOTcshrc` (for `csh`-like shells), `LIBROOT/ciao/DOTprofile` (for `sh`-like shells), and `LIBROOT/ciao/ciao-mode-init` (for emacs) with appropriate definitions which will take care of all needed environment variable definitions and emacs mode setup. If you has indicated in the menu that the startup files must be modified, then the install process will do it for you, otherwise you can modify by hand these files making the following modifications in your startup scripts, so that these files are used (`<v>libroot</v>` must be replaced with the appropriate value):
   - For users a *csh-compatible shell* (`csh`, `tcsh`, ...), add to `~/.cshrc`:

     ```
     if ( -e <v>libroot</v>/ciao/DOTcshrc ) then
        source <v>libroot</v>/ciao/DOTcshrc
     endif
     ```

     Note: while this is recognized by the terminal shell, and therefore by the text-mode Emacs which comes with Mac OS X, the Aqua native Emacs 21 does not recognize that initialization. It is thus necessary, at this moment, to set manually the Ciao shell (ciaosh) and Ciao library location by hand. This can be done from the Ciao menu within Emacs after a Ciao file has been loaded. We believe that the reason is that Mac OS X does not actually consult the per-user initialization files on startup. It should also be possible to put the right initializations in the .emacs file using the `setenv` function of Emacs-lisp, as in

     ```
     (setenv "CIAOLIB" "<v>libroot</v>/ciao")
     ```

     The same can be done for the rest of the variables initialized in `<v>libroot</v>/ciao/DOTcshrc`
   - For users of an *sh-compatible shell* (`sh`, `bash`, ...), the installer will add to `~/.bashrc` the next lines:

     ```
     if [ -f <v>libroot</v>/ciao/DOTprofile ]; then
        . <v>libroot</v>/ciao/DOTprofile
     fi
     ```

     This will set up things so that the Ciao executables are found and you can access the Ciao system manuals using the `info` command. Note that, depending on your shell, *you may have to log out and back in* for the changes to take effect.
   - Also, if you use `emacs` (highly recommended) the install will add the next line to your `~/.emacs` file:

     ```
     (load-file "<v>libroot</v>/ciao/ciao-mode-init.el")
     (if (file-exists-p "<v>libroot</v>/ciao/ciao-mode-init.el")
       (load-file "<v>libroot</v>/ciao/ciao-mode-init.el")
     )
     ```

   If you are installing Ciao globally in a multi-user machine, make sure that you instruct all users to do the same. If you are the system administrator, the previous steps can be done once and for all, and globally for all users by including the lines above in the

central startup scripts (e.g., in Linux `/etc/bashrc`, `/etc/csh.login`, `/etc/csh.cshrc`, `/etc/skel`, `/usr/share/emacs/.../lisp/site-init.pl`, etc.).

7. **Download and install Emacs (highly recommended):** If the (freely available) `emacs` editor is not installed in your system, its installation is *highly recommended* (if you are installing in a multi-user machine, you may want to do it in a general area so that it is available for other users, even if you do not use it yourself). While it is easy to use Ciao with any editor of your choice, the Ciao distribution includes a very powerful *application development environment* which is based on `emacs` and which enables, e.g., source-level debugging, syntax coloring, context-sensitive on-line help, etc.

   The emacs editor (in all its versions: Un*x, Windows, etc.) can be downloaded from, for example, `http://www.emacs.org/`, and also from the many GNU mirror sites worldwide (See `http://www.gnu.org/` for a list), in the `gnu/emacs` and `gnu/windows/emacs` directories. You can find answers to frequently asked questions (FAQ) about `emacs` in general at `http://www.gnu.org/software/emacs/emacs-faq.text` and about the Windows version at `http://www.gnu.org/software/emacs/windows/ntemacs.html` (despite the `ntemacs` name it runs fine also as is on Win9X and Win2000 machines).
8. **Check installation / read documentation:** You may now want to check your installation (see Section 231.3 [Checking for correct installation on Un*x], page 1105) and read the documentation, which is stored in `DOCDIR` (copied from `CIAOSRC/doc/reference`) and can be easily accessed as explained that same section. There are special "getting started" sections at the beginning of the manual.

If you have any problems you may want to check Section 231.8 [Troubleshooting (nasty messages and nifty workarounds)], page 1108.

The system can be *uninstalled* by typing `./ciaosetup uninstall` in the top directory. Configuration should have not changed since installation, so that the same directories are cleaned (i.e. the variables in `SETTINGS` should have the same value as when the install was performed).

231.3 Checking for correct installation on Un*x

If everything has gone well, several applications and tools should be available to a normal user. Try the following while logged in as a *normal user* (important in order to check that permissions are set up correctly):

- Typing `ciao` (or `ciaosh`) should start the typical Prolog-style top-level shell.
- In the top-level shell, Ciao library modules should load correctly. Type for example `use_module(library(dec10_io))` –you should get back a prompt with no errors reported.
- To exit the top level shell, type `halt.` as usual, or ⟨^D⟩.
- Typing `ciaoc` should produce the help message from the Ciao standalone compiler.
- Typing `ciao-shell` should produce a message saying that no code was found. This is a Ciao application which can be used to write scripts written in Ciao, i.e., files which do not need any explicit compilation to be run.

Also, the following documentation-related actions should work:

- If the `info` program is installed, typing `info` should produce a list of manuals which *should include Ciao manual(s) in a separate area* (you may need to log out and back in so that your shell variables are reinitialized for this to work).
- Opening with a WWW browser (e.g., `netscape`) the directory or `URL` corresponding to the `DOCDIR` setting should show a series of Ciao-related manuals. Note that *style sheets* should be activated for correct formatting of the manual.
- Typing `man ciao` should produce a man page with some very basic general information on Ciao (and pointing to the on-line manuals).

- The `DOCDIR` directory should contain the manual also in the other formats such as `postscript` or `pdf` which specially useful for printing. See Section 2.3.7 [Printing manuals (Un*x)], page 24 for instructions.

Finally, if `emacs` is installed, after starting it (typing `emacs`) the following should work:

- Typing ⟨^H⟩ ⟨i⟩ (or in the menus `Help->Manuals->Browse Manuals with Info`) should open a list of manuals in info format in which the Ciao manual(s) should appear.
- When opening a Ciao file, i.e., a file with `.pl` or `.pls` ending, using ⟨^X⟩⟨^F⟩`filename` (or using the menus) the code should appear highlighted according to syntax (e.g., comments in red), and `Ciao/Prolog` menus should appear in the menu bar on top of the `emacs` window.
- Loading the file using the `Ciao/Prolog` menu (or typing ⟨^C⟩ ⟨l⟩) should start in another emacs buffer the Ciao toplevel shell and load the file. You should now be able to switch the the toplevel shell and make queries from within `emacs`.

Note: when using `emacs` it is *very convenient* to swap the locations of the (normally not very useful) ⟨Caps Lock⟩ key and the (very useful in `emacs`) ⟨Ctrl⟩ key on the keyboard. How to do this is explained in the `emacs` frequently asked questions FAQs (see the `emacs` download instructions for their location).

231.4 Cleaning up the source directory

After installation, the source directory can be cleaned up in several ways:

- `./ciaosetup uninstall` removes the installation but does not touch the source directories.
- `./ciaosetup totalclean` leaves the distribution is its original form, throwing away any intermediate files (as well as any unneeded files left behind by the Ciao developers), while still allowing recompilation.

231.5 Multiarchitecture support

As mentioned before, Ciao applications (including the compiler and the top level) can run on several machines with different architectures without any need for recompiling, provided there is one Ciao engine (compiled for the corresponding architecture) accessible in each machine. Also, the Ciao libraries (installed in `LIBROOT`, which contain also the engines) and the actual binaries (installed in `BINROOT`) can themselves be shared on several machines with different architectures, saving disk space.

For example, assume that the compiler is installed as:

`/usr/local/share/bin/ciaoc`

and the libraries are installed under

`/usr/local/share/lib`

Assume also that the `/usr/local/share` directory is mounted on, say, a number of Linux and a number of Solaris boxes. In order for `ciaoc` to run correctly on both types of machines, the following is needed:

1. Make sure you that have done `./ciaosetup install` on one machine of each architecture (once for Linux and once for Solaris in our example). This recompiles and installs a new engine and any architecture-dependent parts of the libraries for each architecture. The engines will have names such as `ciaoengine.LINUXi86`, `ciaoengine.SolarisSparc`, and so on.
2. In multi-architecture environments it is even more important to make sure that users make the modifications to their startup scripts using `<v>libroot</v>/ciao/DOTcshrc` etc. The selection of the engine (and architecture-dependent parts of libraries) is done in these scripts by setting the environment variable `CIAOARCH`, using the `ciao_get_arch` command, which

is installed automatically when installing Ciao. This will set `CIAOARCH` to, say, `LINUXi86`, `SolarisSparc`, respectively, and `CIAOENGINE` will be set to `ciaoengine.`*CIAOARCH*.

However, note that this is not strictly necessary if running on only one architecture: if `CIAOARCH` is not set (i.e., undefined), the Ciao executables will look simply for `ciaoengine`, which is always a link to the latest engine installed in the libraries. But including the initialization files provided has the advantage of setting also paths for the manuals, etc.

231.6 Installation and compilation under Windows

There are two possibilities in order to install Ciao on Windows machines:

- Installing from the Windows *precompiled* distribution. This is the easiest since it requires no compilation and is highly recommended. This is described in Chapter 232 [Installing Ciao from a Win32 binary distribution], page 1111.
- Installing the standard Ciao (Un*x) system source distribution and compiling it under Windows. This is somewhat more complex and currently requires the (freely available) Cygnus Win32 development libraries –described below.

In order to compile Ciao for Win32 environments you need to have the (public domain) *Cygnus Win32* and development libraries installed in your system. Compilation should be performed preferably under Windows NT-type systems.

- Thus, the first step, if Cygnus Win32 is not installed in your system, is to download it (from, e.g., `http://www.cygnus.com/misc/gnu-win32`) and install it. The compilation process also requires that the executables `rm.exe`, `sh.exe`, and `uname.exe` from the Cygnus distribution be copied under `/bin` prior to starting the process (if these executables are not available under `/bin` the compilation process will produce a number of errors and eventually stop prematurely).
- Assuming all of the above is installed, type `./ciaosetup allwin32`. This will compile both the engine and the Ciao libraries. In this process, system libraries that are normally linked dynamically under Un*x (i.e., those for which `.so` dynamically loadable files are generated) are linked statically into the engine (this is done instead of generating `.dll`s because of a limitation in the current version of the Cygnus Win32 environment). No actual installation is made at this point, i.e., this process leaves things in a similar state as if you had just downloaded and uncompressed the precompiled distribution. Thus, in order to complete the installation you should now:
- Follow now the instructions in Chapter 232 [Installing Ciao from a Win32 binary distribution], page 1111.

A further note regarding the executables generated by the Ciao compiler and top-level: the same considerations given in Chapter 232 [Installing Ciao from a Win32 binary distribution], page 1111 apply regarding `.bat` files, etc. However, in a system in which Cygnus Win32 is installed these executables can also be used in a very simple way. In fact, the executables can be run as in Un*x by simply typing their name at the `bash` shell command line without any associated `.bat` files. This only requires that the `bash` shell which comes with Cygnus Win32 be installed and accessible: simply, make sure that `/bin/sh.exe` exists.

231.7 Porting to currently unsupported operating systems

If you would like to port Ciao to a currently unsupported platform, there are several issues to take into account. The main one is to get the *engine* to compile in that platform, i.e., the C code under the `engine` directory. The procedure currently followed by Ciao to decide the various flags needed to compile is as follows:

- The shell script `$(CIAOSRC)/etc/ciao_get_arch` is executed; it returns a string describing the operating system and the processor architecture (e.g., LINUXi86, SolarisSparc, SolarisAlpha, etc.). You should make sure it returns a correct (and meaningful) string for your setup. This string is used trhoughout the compilation to create several architecture-dependant flags.
- In the directory `$(CIAOSRC)/makefile-sysdep` there are files called mkf-<OS><ARCH> for every combination of operating system and architecture in which Ciao is know to (and how to) compile. They set several flags regarding, for example, whether to use or not threads, which threads library to use, the optimization flags to use, the compiler, linker, and it also sets separately the architecture name (ARCHNAME variable) and the operating system (OSNAME). You should create a new mkf file for your machine, starting from the one which is closest to you.
- Most times the porting problems happen in the use of locks and threads. You can either disable them, or have a look at the files `$(CIAOSRC)/engine/locks.h` and `$(CIAOSRC)/engine/threads.h`. If you know how to implement native (assembler) locks for your architecture, enable HAVE_NATIVE_SLOCKS for your architecture and add the definitions. Otherwise, if you have library-based locks, enable them. The mechanism in `threads.h` is similar.

Once a working engine is achieved, it should be possible to continue with the standard installation procedure, which will try to use a completely static version of the standalone compiler (`ciaoc.sta` in the `ciaoc` directory) to compile the interactive top-level (`ciaosh`) and a new version of the standalone compiler (`ciaoc`). These in turn should be able to compile the Ciao libraries. You may also need to look at some libraries (such as, for example, `sockets`) which contain C code. If you do succeed in porting to a platform that is currently unsupported please send the `mkf-CIAOARCH` and any patches to `ciao@clip.dia.fi.upm.es`, and we will include them (with due credit, of course) in the next distribution.

231.8 Troubleshooting (nasty messages and nifty workarounds)

The following a list of common installation problems reported by users:

- **Problem:** Compilation errors appear when trying a new installation/compilation after the previous one was aborted (e.g., because of errors).

  Possible reason and solution: It is a good idea to clean up any leftovers from the previous compilation using `./ciaosetup engclean` before restarting the installation or compilation process.

- **Problem:**

  During engine compilation, messages such as the following appear: `tasks.c:102:PTHREAD_CANCEL_ASYNCHRONOUS undeclared (first use of this function)`.

  Possible reason and solution:

  Your (Linux?) system does not have (yet) the Posix threads library installed. You can upgrade to one which does have it, or download the library from

  `http://pauillac.inria.fr/~xleroy/linuxthreads/index.html`

  and install it, or disable the use of threads in Linux: for this, edit the `SETTINGS` file and specify `USE_THREADS=no`, which will avoid linking against thread libraries (it will disable the use of thread-related primitives as well). Clean the engine with `./ciaosetup engclean` and restart compilation.

  If you have any alternative threads library available, you can tinker with `engine/threads.h` and the files under `makefile-sysdep` in order to get the task managing macros right for your system. Be sure to link the right library. If you succeed, we (`ciao@clip.dia.fi.upm.es`) will be happy of knowing about what you have done.

- **Problem:**

  `-lpthread: library not found` (or similar)

  Possible reason and solution:

  Your (Linux?) system seems to have Posix threads installed, but there is no threads library in the system. In newer releases (e.g., RedHat 5.0), the Posix threads system calls have been included in `glibc.so`, so specifying `-lpthread` in `makefile-sysdep/mkf-LINUX` is not needed; remove it. `./ciaosetup engclean` and restart installation.

  Alternatively, you may have made a custom installation of Posix threads in a non-standard location: be sure to include the flag `-L/this/is/where/the/posix/libraries/are` **before** `-lpthread`, and to update `/etc/ld.so.conf` (see `man ldconfig`).

- **Problem:**

  `Segmentation Violation` (when starting the first executable)

  Possible reason and solution:

  This has been observed with certain older versions of `gcc` which generated erroneous code under full optimization. The best solution is to upgrade to a newer version of `gcc`. Alternatively, lowering the level of optimization (by editing the `SETTINGS` file in the main directory of the distribution) normally solves the problem, at the cost of reduced execution speed.

- **Problem:** `ciaoc: /home/clip/lib/ciao/ciao-X.Y/engine/ciaoengine: not found`

  Possible reason and solution:

  - The system was not fully installed and the variable `CIAOENGINE` was not set.
  - The system was installed, the variable `CIAOENGINE` is set, but it is does not point to a valid ciaoengine.

  See the file `LIBROOT/ciao/DOTcshrc` for user settings for environment variables.

- **Problem:**

  `ERROR: File library(compiler) not found - aborting...` (or any other library is not found)

  Possible reason and solution:

  - The system was not installed and the variable `CIAOLIB` was not set.
  - The system is installed and the variable `CIAOLIB` is wrong.

  See the file `LIBROOT/ciao/DOTcshrc` for user settings for environment variables.

- **Problem:**

  `ERROR: File <some_directory>/<some_file>.itf not found - aborting...`

  Possible reason and solution:

  Can appear when compiling `.pl` files. The file to compile (<some_file>.pl) is not in the directory <some_directory>. You gave a wrong file name or you are in the wrong directory.

- **Problem:**

  `*ERROR*: /(write_option,1) is not a regular type` (and similar ones)

  Possible reason and solution:

  This is not a problem, but rather the type checker catching some minor inconsistencies which may appear while compiling the libraries. Bug us to remove it, but ignore it for now.

- **Problem:**

  `WARNING: Predicate <some_predicate>/<N> undefined in module <some_module>`

  Possible reason and solution:

  It can appear when the compiler is compiling Ciao library modules. If so, ignore it (we will fix it). If it appears when compiling user programs or modules, you may want to check your program for those undefined predicates.

- **Problem:**

  `gmake[1]: execve: /home/clip/mcarro/ciao-0.7p2/etc/collect_modules: No such file or directory`

  Possible reason and solution:

  Check if collect_modules is in $(CIAOSRC)/etc and is executable. If it is not here, your distribution is incorrect: please let us know.

- **Problem:**

  `make: Fatal error in reader: SHARED, line 12: Unexpected end of line seen`

  Possible reason and solution:

  You are using standard Un*x make, not GNU's make implementation (gmake).

- **Problem:**

  `WARNING`s or `ERROR`s while compiling the Ciao libraries during installation.

  Possible reason and solution:

  It is possible that you will see some such errors while compiling the Ciao libraries during installation. This is specially the case if you are installing a Beta or Alpha release of Ciao. These releases (which have "odd" version numbers such as 1.5 or 2.1) are typically snapshots of the development directories, on which many developers are working simultaneously, which may include libraries which have typically not been tested yet as much as the "official" distributions (those with "even" version numbers such as 1.6 or 2.8). Thus, minor warnings may not have been eliminated yet or even errors can sneak in. These warnings and errors should not affect the overall operation of the system (e.g., if you do not use the affected library).

232 Installing Ciao from a Win32 binary distribution

Author(s): Daniel Cabeza, Manuel Carro, Manuel Hermenegildo.

This describes the installation of Ciao after downloading the Windows *binary* (i.e., *precompiled*) distribution. It includes the installation of libraries and manuals and applies to Windows 95/98/NT/2000/XP systems. This is the simplest Windows installation, since it requires no compilation and is highly recommended. However, it is also possible to compile Ciao from the source distribution on these systems (please refer to Chapter 231 [Installing Ciao from the source distribution], page 1101 for details).

If you find any problems during installation, please refer to Section 231.8 [Troubleshooting (nasty messages and nifty workarounds)], page 1108. See also Section 233.3 [Downloading new versions], page 1115 and Section 233.4 [Reporting bugs], page 1116.

232.1 Win32 binary installation summary

Please follow these steps (below we use the terms *folder* and *directory* interchangeably):

1. Download the precompiled distribution and unpack it into any suitable folder, such as, e.g., `C:\Program Files`.

   This will create there a folder whose name reflects the Ciao version. Due to limitations of Windows related to file associations, do not put Ciao too deep in the folder hierarchy. For unpacking you will need a recent version of a zip archive manager – there are many freely available such as `WinZip`, `unzip`, `pkunzip`, etc. (see for example `www.winzip.com`). Some users have reported some problems with version 6.2 of WinZip, but no problems with, e.g., version 7. With WinZip, simply click on "Extract" and select the extraction folder as indicated above.
2. Stop any Ciao-related applications.

   If you have a previous version of Ciao installed, make sure you do not have any Ciao applications (including, e.g., a toplevel shell) running, or the extraction process may not be able to complete. You may also want to delete the entire folder of the previous installation to save space.
3. Open the Ciao source directory created during extraction and run (e.g. by double-clicking on it) the `install(.bat)` script. Answer "yes" to the dialog that pops up and type any character in the installation window to finish the process. You may need to reboot for the changes in the registry to take effect.

   This will update the windows registry (the file `ciao(.reg)` lists the additions) and also create some `.bat` files which may be useful for running Ciao executables from the command line. It also creates initialization scripts for the `emacs` editor. The actions performed by the installation script are reported in the installation window.
4. You may want to add a *windows shortcut* in a convenient place, such as the desktop, to `ciaosh.cpx`, the standard interactive toplevel shell. It is located inside the `shell` folder (e.g., click on the file `ciaosh.cpx` with the right mouse button and select the appropriate option, `Send to->Desktop as shortcut`).
5. You may also want to add another shortcut to the file `ciao(.html)` located inside `doc\reference\ciao_html` so that you can open the Ciao manual by simply double-clicking on this shortcut.
6. Finally, if the (freely available) `emacs` editor/environment is not installed in your system, we *highly recommend* that you also install it at this point. While it is easy to use Ciao with any editor of your choice, the Ciao distribution includes a very powerful *application development environment* which is based on `emacs` and which enables, e.g., source-level debugging, syntax coloring, context-sensitive on-line help, etc. If you are not convinced, consider that many programmers inside Micros*ft use `emacs` for developing their programs.

The emacs editor (in all its versions: Un*x, Windows, etc.) can be downloaded from, for example, `http://www.emacs.org/`, and also from the many GNU mirror sites worldwide (See `http://www.gnu.org/` for a list), in the `gnu/emacs` and `gnu/windows/emacs` directories. You can find answers to frequently asked questions (FAQ) about `emacs` in general at `http://www.gnu.org/software/emacs/emacs-faq.text` and about the Windows version at `http://www.gnu.org/software/emacs/windows/ntemacs.html` (despite the `ntemacs` name it runs fine also as is on Win9X and Win2000 machines).

You need to tell `emacs` how to load the Ciao mode automatically when editing and how to access the on-line documentation:

- Start `emacs` (double click on the icon or from the `Start` menu). Open (menu `Files->Open File` or simply ⟨^X⟩⟨^F⟩) the file `ForEmacs.txt` that the installation script has created in directory where you installed the Ciao distribution.
- Copy the lines in the file (select with the mouse and then menu `Edit->Copy`). Open/Create using `emacs` (menu `Files->Open File` or simply ⟨^X⟩⟨^F⟩) the file `~/.emacs` (or, if this fails, `c:/.emacs`).
- Paste the two lines (menu `Edit->Paste` or simply ⟨^Y⟩) into the file and save (menu `Files->Save Buffer` or simply ⟨^X⟩⟨^S⟩).
- Exit `emacs` and start it again.

`emacs` should not report any errors (at least related to Ciao) on startup. At this point the `emacs` checks in the following section should work.

232.2 Checking for correct installation on Win32

After the actions and registry changes performed by the installation procedure, you should check that the following should work correctly:

- Ciao-related file types (`.pl` source files, `.cpx` executables, `.itf,.po,.asr` interface files, `.pls` scripts, etc.) should have specific icons associated with them (you can look at the files in the folders in the Ciao distribution to check).
- Double-clicking on the shortcut to `ciaosh(.cpx)` on the desktop should start the typical Prolog-style top-level shell in a window. If this shortcut has not been created on the desktop, then double-clicking on the `ciaosh(.cpx)` icon inside the `shell` folder within the Ciao source folder should have the same effect.
- In the top-level shell, Ciao library modules should load correctly. Type for example `use_module(library(dec10_io)).` at the Ciao top-level prompt –you should get back a prompt with no errors reported.
- To exit the top level shell, type `halt.` as usual, or ⟨^D⟩.

Also, the following documentation-related actions should work:

- Double-clicking on the shortcut to `ciao(.html)` which appears on the desktop should show the Ciao manual in your default WWW browser. If this shortcut has not been created you can double-click on the `ciao(.html)` file in the `doc\reference\ciao_html` folder inside the Ciao source folder. Make sure you configure your browser to use *style sheets* for correct formatting of the manual (note, however, that some older versions of Explorer did not support style sheets well and will give better results turning them off).
- The `doc\reference` folder contains the manual also in the other formats present in the distribution, such as `info` (very convenient for users of the `emacs` editor/program development system) and `postscript` or `pdf`, which are specially useful for printing. See Section 3.2.7 [Printing manuals (Win32)], page 29 for instructions.

Finally, if `emacs` is installed, after starting it (double-clicking on the `emacs` icon or from the `Start` menu) the following should work:

- Typing ⟨^H⟩ i (or in the menus `Help->Manuals->Browse Manuals with Info`) should open a list of manuals in info format in which the Ciao manual(s) should appear.
- When opening a Ciao file, i.e., a file with `.pl` or `.pls` ending, using ⟨^X⟩⟨^F⟩`filename` (or using the menus) the code should appear highlighted according to syntax (e.g., comments in red), and `Ciao/Prolog` menus should appear in the menu bar on top of the `emacs` window.
- Loading the file using the `Ciao/Prolog` menu (or typing ⟨^C⟩ l) should start in another emacs buffer the Ciao toplevel shell and load the file. You should now be able to switch the the toplevel shell and make queries from within `emacs`.

Note: when using `emacs` it is *very convenient* to swap the locations of the (normally not very useful) ⟨Caps Lock⟩ key and the (very useful in `emacs`) ⟨Ctrl⟩ key on the keyboard. How to do this is explained in the `emacs` frequently asked questions FAQs (see the `emacs` download instructions for their location).

If you find that everything works but `emacs` cannot start the Ciao toplevel you may want to check if you can open a normal Windows shell within `emacs` (just do ⟨M-x⟩ `shell`). If you cannot, it is possible that you are using some anti-virus software which is causing problems. See `http://www.gnu.org/software/emac/windows/faq3.html#anti-virus` for a workaround.

In some Windows versions it is possible that you had to change the *first* backslashes in the ciao-mode-init.el file in the Ciao Directory. E.g., assuming you have installed in drive `c:`, instances of `c:\` need to be changed to `c:/`. For example: `c:\prolog/ciao-1.7p30Win32/shell/ciaosh.bat` should be changed to `c:/prolog/ciao-1.7p30Win32/shell/ciaosh.bat`.

232.3 Compiling the miscellaneous utilities under Windows

The `etc` folder contains a number of utilities, documented in the manual in *PART V - Miscellaneous Standalone Utilities*. In the Win32 distribution these utilities are not compiled by the installation process. You can create the executable for each of them when needed by compiling the corresponding `.pl` file.

232.4 Server installation under Windows

If you would like to install Ciao on a server machine, used by several clients, the following steps are recommended:

- Follow the standard installation procedure on the server. When selecting the folder in which Ciao is installed make sure you select a folder that is visible by the client machines. Also make sure that the functionality specified in the previous sections is now available on the server.
- Perform a *client installation* on each client, by running (e.g., double-click on it) the `client.bat` script. This should update the registry of each client. At this point all the functionality should also be available on the clients.

232.5 CGI execution under IIS

The standard installation procedure updates the windows registry so that Ciao executables (ending in `.cpx`) are directly executable as CGIs under Microsoft's IIS, i.e., so that you make applications written in Ciao available on the WWW (see the `pillow` library for specific support for this task). In the event you re-install IIS, you probably would lose the entries in the registry which allow this. In that case, processing the file `ciao.reg` produced during the installation (or simply reinstalling Ciao) will add those entries again.

232.6 Uninstallation under Windows

To uninstall Ciao under Windows, simply delete the directory in which you put the Ciao distribution. If you also want to delete the registry entries created by the Ciao installation (not strictly needed) this must currently be done by hand. The installation leaves a list of these entries in the file `ciao.reg` to aid in this task. Also, all the register entries contain the word *ciao*. Thus, to delete all Ciao entries, run the application `regedit` (for example, by selecting `Run` from the Windows `Start` menu), search (⟨F⟩) for *ciao* in all registry entries (i.e., select all of `Keys`, `Values`, and `Data` in the `Edit->Find` dialog), and delete each matching key (click on the left window to find the matching key for each entry found).

%% Local Variables: %% mode: CIAO %% update-version-comments: "off" %% End:

233 Beyond installation

Author(s): Manuel Carro, Daniel Cabeza, Manuel Hermenegildo.

233.1 Architecture-specific notes and limitations

Ciao makes use of advanced characteristics of modern architectures and operating systems such as multithreading, shared memory, sockets, locks, dynamic load libraries, etc., some of which are sometimes not present in a given system and others may be implemented in very different ways across the different systems. As a result, currently not all Ciao features are available in all supported operating systems. Sometimes this is because not all the required features are present in all the OS flavors supported and sometimes because we simply have not had the time to port them yet.

The current state of matters is as follows:

LINUX: multithreading, shared DB access, and locking working.

Solaris: multithreading, shared DB access, and locking working.

IRIX: multithreading, shared DB access, and locking working.

SunOS 4: multithreading, shared DB access, and locking NOT working.

Win 95/98/NT/2000/XP:
multithreading, shared DB access, and locking working. Dynamic linking of object code (C) libraries NOT working.

Mac OS X (Darwin):
multithreading, shared DB access, and locking working.

The features that do not work are disabled at compile time.

233.2 Keeping up to date with the Ciao users mailing list

We recommend that you join the Ciao *users mailing list* (`ciao-users@clip.dia.fi.upm.es`), in order to receive information on new versions and solutions to problems. Simply send a message to `ciao-users-request@clip.dia.fi.upm.es`, containing in the body only the word:

```
subscribe
```

alone in one line. Messages in the list are strictly limited to issues directly related to Ciao and your email address will of course be kept strictly confidential. You mail also want to subscribe to the `comp.lang.prolog` newsgroup.

There is additional info available on the Ciao system, other CLIP group software, publications on the technology underlying these systems, etc. in the CLIP group's WWW site `http://clip.dia.fi.upm.es`.

233.3 Downloading new versions

Ciao and its related libraries and utilities are under constant improvement, so you should make sure that you have the latest versions of the different components, which can be dowloaded from:

```
http://clip.dia.fi.upm.es/Software
```

233.4 Reporting bugs

If you still have problems after downloading the latest version and reading the installation instructions you can send a message to `ciao-bug@clip.dia.fi.upm.es`. Please be as informative as possible in your messages, so that we can reproduce the bug.

- For *installation problems* we typically need to have the version and patch number of the Ciao package (e.g., the name of the file downloaded), the output produced by the installation process (you can capture it by redirecting the output into a file or cutting and pasting with the mouse), and the exact version of the Operating System you are using (as well as the C compiler, if you took a source distribution).
- For *problems during use* we also need the Ciao and OS versions and a small example of code which we can run to reproduce the bug.

References

[**AAF91**] J. Almgren, S. Andersson, L. Flood, C. Frisk, H. Nilsson, and J. Sundberg.
Sicstus Prolog Library Manual.
Po Box 1263, S-16313 Spanga, Sweden, October 1991.

[**AKNL86**] Hassan Ait-Kaci, Roger Nasr, and Pat Lincoln.
E An Overview.
Technical Report AI-420-86-P, Microelectronics and Computer Technology Corporation, 9430 Research Boulevard, Austin, TX 78759, December 1986.

[**AKPS92**] H. Aït-Kaci, A. Podelski, and G. Smolka.
A feature-based constraint system for logic programming with entailment.
In *Proc. Fifth Generation Computer Systems 1992*, pages 1012–1021, 1992.

[**Apt97**] K. Apt, editor.
From Logic Programming to Prolog.
Prentice-Hall, Hemel Hempstead, Hertfordshire, England, 1997.

[**BA82**] M. Ben-Ari.
Principles of Concurrent Programming.
Prentice Hall International, 1982.

[**BBP81**] D.L. Bowen, L. Byrd, L.M. Pereira, F.C.N. Pereira, and D.H.D. Warren.
Decsystem-10 prolog user's manual.
Technical report, Department of Artificial Intelligence, University of Edinburgh, October 1981.

[**BCC97**] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla.
The Ciao Prolog System. Reference Manual.
The Ciao System Documentation Series–TR CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), August 1997.
System and on-line version of the manual available at `http://www.ciaohome.org` (`http://www.ciaohome.org`).

[**BGH99**] F. Bueno, M. García de la Banda, and M. Hermenegildo.
Effectiveness of Abstract Interpretation in Automatic Parallelization: A Case Study in Logic Programming.
ACM Transactions on Programming Languages and Systems, 21(2):189–238, March 1999.

[**BLGPH04**] F. Bueno, P. López-García, G. Puebla, and M. Hermenegildo.
The Ciao Prolog Preprocessor.
Technical Report CLIP1/04, Technical University of Madrid (UPM), Facultad de Informática, 28660 Boadilla del Monte, Madrid, Spain, January 2004.

[**Bue95**] F. Bueno.
The CIAO Multiparadigm Compiler: A User's Manual.
Technical Report CLIP8/95.0, Facultad de Informática, UPM, June 1995.

[**Byr80**] L. Byrd.
Understanding the Control Flow of Prolog Programs.
In S.-A. Tärnlund, editor, *Proceedings of the 1980 Logic Programming Workshop*, pages 127–138, Debrecen, Hungary, July 1980.

[**Cab04**] D. Cabeza.
An Extensible, Global Analysis Friendly Logic Programming System.
PhD thesis, Universidad Politécnica de Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, August 2004.

[**Car87**] M. Carlsson.
Freeze, Indexing, and Other Implementation Issues in the Wam.
In *Fourth International Conference on Logic Programming*, pages 40–58. University of Melbourne, MIT Press, May 1987.

[**Car88**] M. Carlsson.
Sicstus Prolog User's Manual.
Po Box 1263, S-16313 Spanga, Sweden, February 1988.

[**CCG98**] I. Caballero, D. Cabeza, S. Genaim, J.M. Gomez, and M. Hermenegildo.
persdb_sql: SQL Persistent Database Interface.
Technical Report D3.1.M2-A2 CLIP10/98.0, RADIOWEB Project, December 1998.

[**CCH06**] A. Casas, D. Cabeza, and M. Hermenegildo.
A Syntactic Approach to Combining Functional Notation, Lazy Evaluation and Higher-Order in LP Systems.
In *The 8th International Symposium on Functional and Logic Programming (FLOPS'06)*, pages 142–162, Fuji Susono (Japan), April 2006.

[**CGH93**] M. Carro, L. Gómez, and M. Hermenegildo.
Some Paradigms for Visualizing Parallel Execution of Logic Programs.
In *1993 International Conference on Logic Programming*, pages 184–201. MIT Press, June 1993.

[**CH95**] D. Cabeza and M. Hermenegildo.
Distributed Concurrent Constraint Execution in the CIAO System.
In *Proc. of the 1995 COMPULOG-NET Workshop on Parallelism and Implementation Technologies*, Utrecht, NL, September 1995. U. Utrecht / T.U. Madrid.
Available from http://www.cliplab.org/.

[**CH97**] D. Cabeza and M. Hermenegildo.
WWW Programming using Computational Logic Systems (and the PiLLoW/Ciao Library).
In *Proceedings of the Workshop on Logic Programming and the WWW at WWW6*, San Francisco, CA, April 1997.

[**CH99a**] D. Cabeza and M. Hermenegildo.
Higher-order Logic Programming in Ciao.
Technical Report CLIP7/99.0, Facultad de Informática, UPM, September 1999.

[**CH99b**] D. Cabeza and M. Hermenegildo.
The Ciao Modular Compiler and Its Generic Program Processing Library.
In *ICLP'99 WS on Parallelism and Implementation of (C)LP Systems*, pages 147–164. N.M. State U., December 1999.

[**CH00a**] D. Cabeza and M. Hermenegildo.
A New Module System for Prolog.
In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 131–148. Springer-Verlag, July 2000.

[**CH00b**] D. Cabeza and M. Hermenegildo.
The Ciao Modular, Standalone Compiler and Its Generic Program Processing Library.
In *Special Issue on Parallelism and Implementation of (C)LP Systems*, volume 30(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier - North Holland, March 2000.

[**CH00c**] M. Carro and M. Hermenegildo.
Tools for Constraint Visualization: The VIFID/TRIFID Tool.

In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 253–272. Springer-Verlag, September 2000.

[**CH00d**] M. Carro and M. Hermenegildo.
Tools for Search Tree Visualization: The APT Tool.
In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 237–252. Springer-Verlag, September 2000.

[**CHGT98**] D. Cabeza, M. Hermenegildo, S. Genaim, and C. Taboch.
Design of a Generic, Homogeneous Interface to Relational Databases.
Technical Report D3.1.M1-A1, CLIP7/98.0, RADIOWEB Project, September 1998.

[**CHL04**] D. Cabeza, M. Hermenegildo, and J. Lipton.
Hiord: A Type-Free Higher-Order Logic Programming Language with Predicate Abstraction.
In *Ninth Asian Computing Science Conference (ASIAN'04)*, number 3321 in LNCS, pages 93–108. Springer-Verlag, December 2004.

[**CHV96a**] D. Cabeza, M. Hermenegildo, and S. Varma.
The PiLLoW/Ciao Library for INTERNET/WWW Programming using Computational Logic Systems.
In *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, pages 72–90, JICSLP'96, Bonn, September 1996.

[**CHV96b**] D. Cabeza, M. Hermenegildo, and S. Varma.
The **P***i***LL***o***W**/Ciao Library for INTERNET/WWW Programming using Computational Logic Systems.
In *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP'96, Bonn, September 1996.
Available from `http://clement.info.umoncton.ca/~lpnet` (`http://clement.info.umoncton.ca/~lpnet`).

[**CLI95**] The CLIP Group.
CIAO Compiler: Distributed Execution and Low Level Support Subsystem.
Public Software, ACCLAIM Deliverable D4.3/2-A3, Facultad de Informática, UPM, June 1995.

[**CM81**] W.F. Clocksin and C.S. Mellish.
Programming in Prolog.
Springer-Verlag, 1981.

[**Col78**] A. Colmerauer.
Metamorphosis grammars.
In *Natural language communication with computers*, pages 133–189. Springer LNCS 63, 1978.

[**Col82**] A. Colmerauer et al.
Prolog II: Reference Manual and Theoretical Model.
Groupe D'intelligence Artificielle, Faculté Des Sciences De Luminy, Marseille, 1982.

[**DEDC96**] P. Deransart, A. Ed-Dbali, and L. Cervoni.
Prolog: The Standard.
Springer-Verlag, 1996.

[**Dij65**] E.W. Dijkstra.
Co-operating sequential processes.
In F. Genuys, editor, *Programming Languages.* Academic Press, London, 1965.

[**DL93**] S. K. Debray and N. W. Lin.
Cost Analysis of Logic Programs.
ACM Transactions on Programming Languages and Systems, 15(5):826–875, November 1993.

[**DLGH97**] S.K. Debray, P. López-García, and M. Hermenegildo.
Non-Failure Analysis for Logic Programs.
In *1997 International Conference on Logic Programming*, pages 48–62, Cambridge, MA, June 1997. MIT Press, Cambridge, MA.

[**DLGHL97**] S. K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin.
Lower Bound Cost Estimation for Logic Programs.
In *1997 International Logic Programming Symposium*, pages 291–305. MIT Press, Cambridge, MA, October 1997.

[**GCH98**] J.M. Gomez, D. Cabeza, and M. Hermenegildo.
WebDB: A Database WWW Interface.
Technical Report D3.1.M2-A3 CLIP11/98.0, RADIOWEB Project, December 1998.

[**GdW94**] J.P. Gallagher and D.A. de Waal.
Fast and precise regular approximations of logic programs.
In Pascal Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming*, pages 599–613. MIT Press, 1994.

[**HBC96**] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla.
The CIAO Multi-Dialect Compiler and System: A Demo and Status Report.
In *Proceedings of the JICSLP'96 Workshop on Parallelism and Implementation Technology*. Computer Science Department, Technical University of Madrid, September 1996.
Available from `http://www.cliplab.org/Projects/COMPULOG/meeting96/papers/PS/clip.ps.gz` (`http://www.cliplab.org/Projects/COMPULOG/meeting96/papers/PS/clip.ps.gz`).

[**HBC99**] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla.
The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems.
In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, Commack, NY, USA, April 1999.

[**HBdlBP95**] M. Hermenegildo, F. Bueno, M. García de la Banda, and G. Puebla.
The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems.
In *Proceedings of the ILPS'95 Workshop on Visions for the Future of Logic Programming*, Portland, Oregon, USA, December 1995.
Available from `http://www.cliplab.org/` (`http://www.cliplab.org/`).

[**HBPLG99**] M. Hermenegildo, F. Bueno, G. Puebla, and P. López-García.
Program Analysis, Debugging and Optimization Using the Ciao System Preprocessor.
In *1999 Int'l. Conference on Logic Programming*, pages 52–66, Cambridge, MA, November 1999. MIT Press.

[**HC93**] M. Hermenegildo and The CLIP Group.
Towards CIAO-Prolog – A Parallel Concurrent Constraint System.

In *Proc. of the Compulog Net Area Workshop on Parallelism and Implementation Technologies*. FIM/UPM, Madrid, Spain, June 1993.

[**HC94**] M. Hermenegildo and The CLIP Group.
Some Methodological Issues in the Design of CIAO - A Generic, Parallel, Concurrent Constraint System.
In *Principles and Practice of Constraint Programming*, number 874 in LNCS, pages 123–133. Springer-Verlag, May 1994.

[**HC97**] M. Hermenegildo and The CLIP Group.
An Automatic Documentation Generator for (C)LP – Reference Manual.
The Ciao System Documentation Series–TR CLIP5/97.3, Facultad de Informática, UPM, August 1997.
Online at `http://www.ciaohome.org`.

[**HCC95**] M. Hermenegildo, D. Cabeza, and M. Carro.
Using Attributed Variables in the Implementation of Concurrent and Parallel Logic Programming Systems.
In *Proc. of the Twelfth International Conference on Logic Programming*, pages 631–645. MIT Press, June 1995.

[**Her86**] M. Hermenegildo.
An Abstract Machine for Restricted AND-parallel Execution of Logic Programs.
In *Third International Conference on Logic Programming*, number 225 in Lecture Notes in Computer Science, pages 25–40. Imperial College, Springer-Verlag, July 1986.

[**Her96**] M. Hermenegildo.
Writing "Shell Scripts" in SICStus Prolog, April 1996.
Posting in `comp.lang.prolog`. Available from `http://www.cliplab.org/` (`http://www.cliplab.org/`).

[**Her99**] M. Hermenegildo.
A Documentation Generator for Logic Programming Systems.
Technical Report CLIP10/99.0, Facultad de Informática, UPM, September 1999.

[**Her00**] M. Hermenegildo.
A Documentation Generator for (C)LP Systems.
In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 1345–1361. Springer-Verlag, July 2000.

[**HG91**] M. Hermenegildo and K. Greene.
The &-Prolog System: Exploiting Independent And-Parallelism.
New Generation Computing, 9(3,4):233–257, 1991.

[**Hog84**] C. J. Hogger.
Introduction to Logic Programming.
Academic Press, London, 1984.

[**Hol90**] C. Holzbaur.
Specification of Constraint Based Inference Mechanisms through Extended Unification.
PhD thesis, University of Vienna, 1990.

[**Hol92**] C. Holzbaur.
Metastructures vs. Attributed Variables in the Context of Extensible Unification.
In *1992 International Symposium on Programming Language Implementation and Logic Programming*, pages 260–268. LNCS631, Springer Verlag, August 1992.

[**Hol94**] C. Holzbaur.
SICStus 2.1/DMCAI Clp 2.1.1 User's Manual.
University of Vienna, 1994.

[**HPBLG05**] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García.
Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor).
Science of Computer Programming, 58(1–2):115–140, 2005.

[**JL88**] D. Jacobs and A. Langen.
Compilation of Logic Programs for Restricted And-Parallelism.
In *European Symposium on Programming*, pages 284–297, 1988.

[**Knu84**] D. Knuth.
Literate programming.
Computer Journal, 27:97–111, 1984.

[**Kor85**] R. Korf.
Depth-first iterative deepening: an optimal admissible tree search.
Artificial Intelligence, 27:97–109, 1985.

[**LGHD96**] P. López-García, M. Hermenegildo, and S. K. Debray.
A Methodology for Granularity Based Control of Parallelism in Logic Programs.
Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation, 21(4–6):715–734, 1996.

[**MH89**] K. Muthukumar and M. Hermenegildo.
Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.
In *1989 North American Conference on Logic Programming*, pages 166–189. MIT Press, October 1989.

[**Nai85**] L. Naish.
The MU-Prolog 3.2 Reference Manual.
TR 85/11, Dept. of Computer Science, U. of Melbourne, October 1985.

[**Nai91**] Lee Naish.
Adding equations to NU-Prolog.
In *Proceedings of The Third International Symposium on Programming Language Implementation and Logic Programming (PLILP'91)*, number 528 in Lecture Notes in Computer Science, pages 15–26, Passau, Germany, August 1991. Springer-Verlag.

[**Par97**] The RADIOWEB Project Partners.
RADIOWEB EP25562: Automatic Generation of Web Sites for the Radio Brodcasting Industry – Project Description / Technical Annex.
Technical Report, RADIOWEB Project, July 1997.

[**PBH97**] G. Puebla, F. Bueno, and M. Hermenegildo.
An Assertion Language for Debugging of Constraint Logic Programs.
In *Proceedings of the ILPS'97 Workshop on Tools and Environments for (Constraint) Logic Programming*, October 1997.
Available from `ftp://clip.dia.fi.upm.es/pub/papers/assert_lang_tr_discipldeliv.ps.gz` (`ftp://clip.dia.fi.upm.es/pub/papers/assert_lang_tr_discipldeliv.ps.gz`) as technical report CLIP2/97.1.

[**PBH00**] G. Puebla, F. Bueno, and M. Hermenegildo.
An Assertion Language for Constraint Logic Programs.
In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.

[**PH99**] G. Puebla and M. Hermenegildo.
Some Issues in Analysis and Specialization of Modular Ciao-Prolog Programs.
In *ICLP'99 Workshop on Optimization and Implementation of Declarative Languages*, pages 45–61. U. of Southampton, U.K, November 1999.

[**PW80**] F.C.N. Pereira and D.H.D. Warren.
Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks.
Artificial Intelligence, 13:231–278, 1980.

[**SS86**] L. Sterling and E. Shapiro.
The Art of Prolog.
MIT Press, 1986.

[**Swe95**] Swedish Institute of Computer Science, P.O. Box 1263, S-16313 Spanga, Sweden.
Sicstus Prolog V3.0 User's Manual, 1995.

[**War88**] D.H.D. Warren.
The Andorra Model.
Presented at Gigalips Project workshop. U. of Manchester, March 1988.

Library/Module Index

A

actmods ... 585
agent ... 589
aggregates ... 233
andorra ... 539
andprolog ... 543
apll ... 545
argnames ... 521
arithmetic ... 151
arrays ... 775
assertions ... 339
assertions_props ... 349
assoc ... 777
assrt_write ... 455
atom2term ... 413
atomic_basic ... 141
attributes ... 201

B

basic_props ... 105
basiccontrol ... 99
basicmodes ... 373
between ... 299
BeyondInstall ... 1115
bf ... 593
block ... 845
bltclass ... 857
boundary ... 1005
build_foreign_interface ... 669
builtin_directives ... 103

C

callgraph ... 825
chartlib ... 847
chartlib_errhandle ... 859
CiaoMode ... 67
ciaopaths ... 383
class ... 617
clpq ... 597
clpr ... 601
color_pattern ... 861
compiler ... 297
conc_aggregates ... 471
concurrency ... 465
condcomp ... 209
counters ... 785
ctrlcclean ... 417
cyclic_terms ... 449

D

data_facts ... 187
davinci ... 685
dcg ... 271
dcg_tr ... 275
ddlist ... 919
debugger ... 49, 59
dec10_io ... 329
default_predicates ... 211
det_hook ... 571
det_hook_rt ... 575
dht_client ... 929
dht_config ... 975
dht_logic ... 941
dht_logic_misc ... 957
dht_misc ... 979
dht_routing ... 953
dht_rpr ... 961
dht_s2c ... 937
dht_s2s ... 939
dht_server ... 935
dht_storage ... 971
dict ... 399
dictionary ... 1007
dictionary_tree ... 1009
dynamic_rt ... 239

E

ecrc ... 385
emacs ... 767
errhandle ... 419
exceptions ... 177
expansion_tools ... 461

F

factsdb ... 733
factsdb_rt ... 735
fastrw ... 421
fd ... 981
field_type ... 1015
field_value ... 1017
field_value_check ... 1019
file_locks ... 433
file_utils ... 429
filenames ... 423
foreign_compilation ... 667
foreign_interface ... 645
foreign_interface_properties ... 661

format ... 277
formulae ... 435
freeze ... 579
fsyntax ... 525
fuzzy ... 605

G

genbar1 ... 865
genbar2 ... 871
genbar3 ... 875
genbar4 ... 879
gendot ... 987
generator ... 1021
generator_util ... 1023
gengraph1 ... 883
gengraph2 ... 893
genmultibar ... 901
getopts ... 391
GetStartUnix ... 21
GetStartWin32 ... 27
global ... 537
gnuplot ... 989
graphs ... 793

H

hiord_rt ... 515
hiordlib ... 517
html ... 703
http ... 713

I

id ... 595
idlists ... 787
indexer ... 511
Install ... 1101
InstallWin32bin ... 1111
interface ... 641
internal_types ... 1035
io_alias_redirection ... 411
io_aux ... 197
io_basic ... 169
iso ... 231
iso_byte_char ... 259
iso_incomplete ... 267
iso_misc ... 265
isomodes ... 371

J

javart ... 749
javasock ... 763
jtopl ... 759

L

lazy ... 993
lgraphs ... 803
libpaths ... 65
librowser ... 457
linda ... 771
lists ... 283
llists ... 395
loading_code ... 97
lookup ... 1041

M

make ... 481
make_rt ... 487
menu ... 673
menu_generator ... 675
messages ... 405
modules ... 91
mycin ... 997

N

native_props ... 359
numlists ... 789

O

objects ... 629
objects_rt ... 635
ociao ... 611
odd ... 577
old_database ... 331
operators ... 255

P

parser_util ... 1049
patterns ... 791
persdbrt ... 723
persdbtr_sql ... 743
pillow ... 701
pillow_types ... 715
pl2sqlinsert ... 745
possible ... 1063
pretty_print ... 451
profiler ... 999
prolog_flags ... 181
prolog_sys ... 323
provrml ... 1001
provrml_io ... 1039
provrml_parser ... 1047
provrmlerror ... 1013
pure ... 509

Q

queues ... 805

R

random ... 807
read ... 245
regexp ... 1069
regexp_code ... 1073
regtypes ... 355
rtchecks ... 377
runtime_ops ... 335

S

sets ... 809
sockets ... 473
sockets_io ... 479
sort ... 295
sqltypes ... 739
streams ... 397
streams_basic ... 159
strings ... 401
symfnames ... 427
syntax_extensions ... 193
system ... 301
system_extra ... 493
system_info ... 205

T

table_widget1 ... 905
table_widget2 ... 909
table_widget3 ... 911
table_widget4 ... 913
tcltk ... 689
tcltk_low_level ... 697
term_basic ... 129
term_compare ... 135
term_typing ... 121
terms ... 441
terms_check ... 445
terms_vars ... 447
test_format ... 915
tester ... 1077
time_analyzer ... 1083
tokeniser ... 1065
toplevel ... 41
ttyout ... 333

U

ugraphs ... 797
unittest ... 379

V

vndict ... 813

W

wgraphs ... 801
when ... 581
write ... 249

X

xdr_handle ... 1089
xml_path ... 1093

Predicate/Method Index

!

!/0 100

$

$~/3 522
$cancellation/1 551
$clean_measures/0 559
$enter_mutex/1 558
$enter_mutex_remote/1 558
$enter_mutex_self/0 558
$exit_mutex/1 559
$exit_mutex_remote/1 559
$exit_mutex_self/0 559
$factsdb$cached_goal/3 ..... 733, 736
$find_goal/3 551
$goal_available/1 551
$goal_cancelled/1 555
$goal_det/1 552
$goal_failed/1 555
$goal_finished/1 553
$goal_not_executed/1 552
$goal_rem_executing/1 553
$goal_tobacktrack/1 554
$goal_toreexecute/1 554
$incr_num_local_backtr/0 560
$is_persistent/2 ..... 681, 729
$meta_call/1 516
$more_solutions/1 556
$move_execution_top/1 557
$new_measure/0 560
$nodebug_call/1 516
$not_measure/0 560
$number_agents/1 550
$print_measures/0 559
$push_goal/3 551
$read_event/1 556
$release/1 557
$release_all_for_unwinding/0 558
$release_remote/1 557
$release_some_suspended_thread/0 558
$retrieve_goal/2 552
$save_end_execution/1 556
$save_init_execution/1 556
$send_event/1 556
$set_goal_cancelled/1 555
$set_goal_det/1 552
$set_goal_failed/1 555
$set_goal_finished/1 554
$set_goal_nondet/1 552
$set_goal_not_executed/1 553
$set_goal_rem_executing/1 553
$set_goal_tobacktrack/1 554
$set_goal_toreexecute/1 554
$show_handler/1 555
$start_thread/1 550
$suspend/0 557
$waiting/1 557

,

,/2 99

-

--/1 496
-/1 496
->/2 100

.

./2 45

:

:#/2 606
::/2 590
:~/2 607

;

;/2 99

=

=../2 132
=:=/2 155
=>/4 608
=\=/2 156
=</2 153

@

@=</2 137
@>/2 137
@>=/2 137
@</2 136

>

>/2 154
>=/2 155

^

^/2 236

\

\=/2 129
\==/2 136
\+/1 100

<

</2 153

A

abolish/1 242
abort/0 179
absolute_file_name/2 164
absolute_file_name/7 165
accepted_type/2 739
acyclic_term/1 449
add_after/4 ..... 290, 787
add_assoc/4 783
add_before/4 ..... 290, 788
add_edges/3 798
add_environment_whitespace/3 1055
add_indentation/3 1056
add_lines/4 200
add_name_value/2 489
add_preffix/3 498
add_suffix/3 498
add_vertices/3 798
add_vpath/1 490
add_vpath_mode/3 490
alias_file/1 428
all_values/2 488
any_to_term/2 499
append/2 395
append/3 283
apply_vpath_mode/4 491
apropos/1 459
aref/3 775
arefa/3 775
arefl/3 776
arg/2 442
arg/3 130
arg_expander/6 462
arithm_average/2 390
array_to_list/2 776
asbody_to_conj/2 438
aset/4 776
ask/2 445
assert/1 240
assert/2 241
asserta/1 239
asserta/2 240
asserta_fact/1 ..... 187, 726, 735
asserta_fact/2 187
assertz/1 240
assertz/2 240
assertz_fact/1 ..... 188, 727, 735
assertz_fact/2 188
assoc_to_list/2 777
at_least_one/4 1049
at_least_one/5 1049
atom_chars/2 259
atom_codes/2 142
atom_concat/2 442
atom_concat/3 147
atom_length/2 147
atom_lock_state/2 469
atom_number/2 143
atom_number/3 146
atom2term/2 413
attach_attribute/2 201

B

bagof/3 ..... 234, 472
barchart1/7 865
barchart1/9 866
barchart2/11 872
barchart2/7 871
barchart3/7 875
barchart3/9 876
barchart4/11 880
barchart4/7 879
basename/2 425
benchmark/6 1084
benchmark2/6 1085
between/3 299
bind_socket/3 474
bind_socket_interface/1 763
body_expander/6 461
body2list/2 438

bold_message/1 490
bold_message/2 490
bolder_message/1 490
bolder_message/2 490
boundary_check/3 1005
boundary_rotation_first/2 1005
boundary_rotation_last/2 1006
bounds/3 985
browse/2 458
build_foreign_interface/1 669
build_foreign_interface_explicit_decls/2 .. 670
build_foreign_interface_object/1 670

C

C/3 133
c_errno/1 305
call/1 ..... 515, 736
call/2 515
call_graph/2 825
call_in_module/2 59
call_unknown/1 488
cancellation/1 562
case_insensitive_match/2 791
cat/2 494
cat_append/2 494
catch/3 177
cd/1 310
char_code/2 259
char_codes/2 260
character_count/2 162
chartlib_text_error_protect/1 859
chartlib_visual_error_protect/1 859
check/1 346
check_sublist/4 916
check_var_exists/1 489
children_nodes/1 1006
chmod/2 317
chmod/3 317
choose_free_var/2 984
choose_value/2 984
choose_var/3 983
choose_var_nd/2 984
ciao_c_headers_dir/1 207
ciao_flag/3 184
ciao_lib_dir/1 208
cl_option/2 392
clause/2 242
clean_measures/0 568
clearerr/1 163
close/1 160
close/2 267
close_client/0 771
close_DEF/5 1032
close_EXTERNPROTO/6 1031
close_file/1 330
close_input/1 397
close_node/5 1029
close_nodeGut/4 1030
close_output/1 398
close_predicate/1 190
close_PROTO/6 1030
close_Script/5 1033
code_class/2 173
collect_singletons/2 395
color/2 862
combine_attributes/2 202
compare/3 138
compare_benchmark/7 1084
compare_benchmark2/7 1085
compile/1 45
compiler_and_opts/2 667
complete_dict/3 814
complete_vars_dict/3 814
compound/1 265
concurrent/1 469
conj_to_list/2 436
conj_to_llist/2 437
connect_to_socket/3 474
connect_to_socket_type/4 473
consistent_hash/2 958
constructor/0 621
consult/1 45
contains_ro/2 291
contains1/2 291
continue/3 1063
convert_atoms_to_string/2 1039
convert_permissions/2 499
convert_permissions/4 499
copy_args/3 441
copy_file/2 305
copy_file/3 305
copy_files/2 494
copy_files/3 494
copy_files_nofail/3 494
copy_stdout/1 429
copy_term/2 132
copy_term_nat/2 133
core/1 700
correct_commenting/4 1052

cost/3 ... 1086
cp_name_value/2 ... 489
create/2 ... 728
create_dict/2 ... 813
create_dictionaries/1 ... 1009
create_directed_field/5 ... 1052
create_environment/4 ... 1054
create_field/3 ... 1051
create_field/4 ... 1051
create_field/5 ... 1051
create_from_list/2 ... 919
create_node/3 ... 1051
create_parse_structure/1 ... 1053
create_parse_structure/2 ... 1053
create_parse_structure/3 ... 1053
create_pretty_dict/2 ... 813
create_proto_element/3 ... 1041
cross_product/2 ... 294
ctrlc_clean/1 ... 417
ctrlcclean/0 ... 417
current_atom/1 ... 327
current_ciao_flag/2 ... 184
current_env/2 ... 304
current_executable/1 ... 308
current_fact/1 ... 188, 736
current_fact/2 ... 189
current_fact_nb/1 ... 190
current_host/1 ... 308
current_infixop/4 ... 256
current_input/1 ... 161
current_key/2 ... 332
current_module/1 ... 207
current_op/3 ... 256
current_output/1 ... 162
current_postfixop/3 ... 257
current_predicate/1 ... 242
current_predicate/2 ... 243
current_prefixop/3 ... 256
current_prolog_flag/2 ... 182
current_stream/3 ... 163
cyclic_term/1 ... 449
cyg2win/3 ... 319

D

data/1 ... 243
datime/1 ... 302
datime/9 ... 302
datime_atom/1 ... 498
datime_atom/2 ... 498
datime_string/1 ... 498
datime_string/2 ... 498
davinci/0 ... 685
davinci_get/1 ... 685
davinci_get_all/1 ... 685
davinci_lgraph/1 ... 686
davinci_put/1 ... 686
davinci_quit/0 ... 686
davinci_ugraph/1 ... 686
dbId/2 ... 743
dcg_translation/2 ... 275
ddlist_member/2 ... 923
debug/1 ... 199
debug_goal/2 ... 408
debug_goal/3 ... 409
debug_message/1 ... 408
debug_message/2 ... 408
dec_indentation/2 ... 1056
decompose_field/3 ... 1033
define_flag/3 ... 61, 186, 247, 254, 320, 711, 1076
del_assoc/4 ... 783
del_dir_if_empty/1 ... 493
del_edges/3 ... 798
del_endings_nofail/2 ... 495
del_env/1 ... 305
del_file_nofail/1 ... 495
del_file_nofail/2 ... 495
del_files_nofail/1 ... 495
del_global/1 ... 537
del_max_assoc/4 ... 784
del_min_assoc/4 ... 784
del_name_value/1 ... 489
del_vertices/3 ... 798
delete/1 ... 700
delete/2 ... 921
delete/3 ... 287, 788
delete_after/2 ... 921
delete_directory/1 ... 318
delete_file/1 ... 317
delete_files/1 ... 495
delete_non_ground/3 ... 287
delete_on_ctrlc/2 ... 417
delete_top/2 ... 921
derived_from/2 ... 637
describe/1 ... 459
destroy/1 ... 638
destructor/0 ... 622
det_try/3 ... 575
detach_attribute/1 ... 202
dgraph_to_ugraph/2 ... 793

dht_check_predecessor/1 ... 942
dht_closest_preceding_finger/2 ... 942
dht_connect/2 ... 929
dht_connect/3 ... 929
dht_consult_b/2 ... 972
dht_consult_b/4 ... 930
dht_consult_nb/2 ... 973
dht_consult_nb/4 ... 931
dht_consult_server_b/3 ... 946
dht_consult_server_nb/3 ... 948
dht_disconnect/1 ... 930
dht_extract_b/2 ... 971
dht_extract_b/4 ... 931
dht_extract_from_server_b/3 ... 949
dht_extract_from_server_nb/3 ... 950
dht_extract_nb/2 ... 972
dht_extract_nb/4 ... 931
dht_find_and_consult_b/2 ... 946
dht_find_and_consult_nb/2 ... 947
dht_find_and_extract_b/2 ... 948
dht_find_and_extract_nb/2 ... 950
dht_find_and_store/2 ... 951
dht_find_predecessor/2 ... 943
dht_find_successor/2 ... 944
dht_finger/2 ... 941
dht_finger_start/2 ... 954
dht_finger_table/2 ... 953
dht_fix_fingers/0 ... 945
dht_hash/3 ... 932
dht_id_by_node/2 ... 946
dht_init/1 ... 941
dht_join/1 ... 944
dht_join_host/1 ... 976
dht_key_hash/2 ... 973
dht_notify/1 ... 944
dht_predecessor/1 ... 956
dht_prolog/1 ... 935
dht_reset_predecessor/0 ... 956
dht_rpr_call/2 ... 965
dht_rpr_call/3 ... 968
dht_rpr_clear_by_node/1 ... 964
dht_rpr_compose_id/3 ... 964
dht_rpr_id_by_node/2 ... 962
dht_rpr_node/1 ... 965
dht_rpr_node_by_id/2 ... 962
dht_rpr_register_node/1 ... 961
dht_rpr_register_node/2 ... 961
dht_s2c_main/0 ... 937
dht_s2c_port/1 ... 975
dht_s2c_threads/1 ... 976
dht_s2s_main/0 ... 939
dht_s2s_port/1 ... 976
dht_s2s_threads/1 ... 976
dht_server/1 ... 935
dht_server_host/1 ... 977
dht_server_id/1 ... 977
dht_set_finger/4 ... 955
dht_set_hash_power/1 ... 975
dht_set_join_host/1 ... 977
dht_set_predecessor/1 ... 956
dht_set_s2c_port/1 ... 976
dht_set_s2c_threads/1 ... 976
dht_set_s2s_port/1 ... 976
dht_set_s2s_threads/1 ... 976
dht_set_server_host/1 ... 977
dht_set_server_id/1 ... 977
dht_stabilize/0 ... 945
dht_store/3 ... 971
dht_store/4 ... 932
dht_store_to_server/4 ... 951
dht_successor/1 ... 942
dht_update_finger/2 ... 955
dic_get/3 ... 400
dic_lookup/3 ... 399
dic_lookup/4 ... 400
dic_node/2 ... 399
dic_replace/4 ... 400
dict2varnamesl/2 ... 814
dictionary/5 ... 399
dictionary/6 ... 1007
dictionary_insert/5 ... 1010
dictionary_lookup/5 ... 1010
diff_vars/3 ... 447
difference/3 ... 293
dir_path/2 ... 305
directory_files/2 ... 313
disj_to_list/2 ... 437
disj_to_llist/2 ... 438
display/1 ... 175
display/2 ... 174
display_list/1 ... 199
display_string/1 ... 199
display_term/1 ... 199
displayq/1 ... 175
displayq/2 ... 175
div_times/2 ... 1086
dlgraph_to_lgraph/2 ... 794
dlist/3 ... 290
do/2 ... 496
do/3 ... 496

do/4 496
do/5 497
do_atmlist__popen/2 500
do_interface/1 671
do_on_abolish/1 244
do_str/3 500
do_str_without_nl/3 500
do_str_without_nl__popen/2 500
dot_concat/2 488
dyn_load_cfg_module_into_make/1 491
dynamic/1 243
dynamic_search_path/1 46

E

edges/2 797
edges_to_lgraph/2 794
edges_to_ugraph/2 794
emacs_edit/1 768
emacs_edit_nowait/1 768
emacs_eval/1 768
emacs_eval_nowait/1 768
empty_assoc/1 777
eng_backtrack/2 466
eng_call/3 466
eng_call/4 465
eng_cut/1 466
eng_goal_id/1 468
eng_kill/1 467
eng_killothers/0 467
eng_release/1 467
eng_self/1 467
eng_status/0 468
eng_wait/1 467
ensure_loaded/1 ..... 44, 297
ensure_loaded/2 297
enter_mutex/1 567
enter_mutex_remote/1 568
enter_mutex_self/0 568
eq/3 681
equal_lists/2 293
equalnumber/3 915
erase/1 ..... 191, 244
error/1 198
error_file/2 860
error_message/1 405
error_message/2 ..... 405, 860
error_message/3 405
error_protect/1 419
error_vrml/1 1013
etags/2 499
exec/3 312
exec/4 312
exec/8 312
execute_permissions/2 499
execute_permissions/4 499
exit_mutex/1 568
exit_mutex_remote/1 568
exit_mutex_self/0 568
extension/2 425
extract_paths/2 306

F

fail/0 101
false/0 102
false/1 347
fast_read/1 421
fast_read/2 421
fast_write/1 421
fast_write/2 421
fast_write_to_string/3 422
fetch_url/3 713
fieldType/1 1015
fieldValue/6 1017
fieldValue_check/8 1019
file_alias/2 ..... 428, 737
file_dir_name/3 306
file_directory_base_name/3 424
file_exists/1 314
file_exists/2 314
file_name_extension/3 424
file_properties/6 315
file_property/2 315
file_search_path/2 ..... 65, 167
file_terms/2 429
file_to_string/2 429
file_to_string/3 430
fileerrors/0 185
fillout/4 1050
fillout/5 1050
filter_alist_pattern/3 495
find_det_goal/2 561
find_file/2 489
find_goal/2 561
find_name/4 815
findall/3 ..... 234, 471
findall/4 235
findnsols/4 235
findnsols/5 236

flatten/2 ... 395
flush_output/0 ... 163
flush_output/1 ... 163
fmode/2 ... 316
fnot/1 ... 607
foldl/4 ... 518, 782
force_lazy/1 ... 46
form_default/3 ... 706
form_empty_value/1 ... 706
form_request_method/1 ... 710
format/2 ... 277
format/3 ... 278
format_to_string/3 ... 278
formatting/2 ... 686
forward/2 ... 922
freeze/2 ... 579
frozen/2 ... 579
functor/3 ... 131
fuzzy/1 ... 606
fuzzy_predicate/1 ... 606

G

garbage_collect/0 ... 327
gc/0 ... 185
gen_assoc/3 ... 778
gendot/3 ... 987
generate_benchmark_list/7 ... 1085
generate_benchmark_list2/7 ... 1086
generate_human_file/0 ... 389
generate_js_menu/1 ... 678
generate_machine_file/0 ... 389
generate_plot/2 ... 990
generate_plot/3 ... 990
generator/2 ... 1021
geom_average/2 ... 390
get_active_config/1 ... 491
get_address/2 ... 308
get_alias_path/0 ... 65
get_all_values/2 ... 488
get_arch/1 ... 205
get_assoc/3 ... 779
get_assoc/5 ... 779
get_attribute/2 ... 201
get_byte/1 ... 261
get_byte/2 ... 261
get_char/1 ... 262
get_char/2 ... 262
get_ciao_ext/1 ... 206
get_code/1 ... 169
get_code/2 ... 169
get_cookies/1 ... 707
get_debug/1 ... 206
get_definition_dictionary/2 ... 1009
get_dictionaries/2 ... 1058
get_eng_location/1 ... 206
get_environment/2 ... 1058
get_environment_name/2 ... 1054
get_environment_type/2 ... 1055
get_exec_ext/1 ... 206
get_first_parsed/3 ... 1061
get_form_input/1 ... 706
get_form_value/3 ... 706
get_general_options/1 ... 989
get_gid/1 ... 307
get_global/2 ... 537
get_grnam/1 ... 307
get_indentation/2 ... 1056
get_line/1 ... 401
get_line/2 ... 401
get_menu_configs/1 ... 676
get_menu_flag/3 ... 676
get_menu_flags/1 ... 677
get_menu_flags/2 ... 678
get_menu_options/2 ... 677
get_name/2 ... 492
get_name_value/3 ... 489
get_name_value_string/3 ... 489
get_next_assoc/4 ... 780
get_os/1 ... 205
get_parsed/2 ... 1057
get_perms/2 ... 499
get_pid/1 ... 306
get_platform/1 ... 206
get_prev_assoc/4 ... 780
get_primes/2 ... 789
get_prototype_definition/2 ... 1042
get_prototype_dictionary/2 ... 1010
get_prototype_interface/2 ... 1041
get_pwnam/1 ... 307
get_row_number/2 ... 1055
get_settings_nvalue/1 ... 491
get_so_ext/1 ... 207
get_stream/2 ... 411
get_tmp_dir/1 ... 307
get_type/2 ... 739
get_uid/1 ... 307
get_value/2 ... 488
get_value_def/3 ... 488
get1_code/1 ... 170

get1_code/2 ... 170
getcounter/2 ... 785
getct/2 ... 173
getct1/2 ... 174
getenvstr/2 ... 304
getopts/4 ... 391
glb/2 ... 985
goal_available/1 ... 561
goal_cancelled/1 ... 565
goal_det/1 ... 562
goal_failed/1 ... 564
goal_finished/1 ... 563
goal_id/1 ... 468
goal_not_executed/1 ... 563
goal_rem_executing/1 ... 563
goal_tobacktrack/1 ... 564
goal_toreexecute/1 ... 564
graph_b1/13 ... 885
graph_b1/9 ... 884
graph_b2/13 ... 894
graph_b2/9 ... 893
graph_w1/13 ... 886
graph_w1/9 ... 885
graph_w2/13 ... 896
graph_w2/9 ... 895

H

halt/0 ... 179
halt/1 ... 179
halt_server/0 ... 772
handle_error/2 ... 419
hash_power/1 ... 975
hash_size/1 ... 957
hash_term/2 ... 511
highest_hash_number/1 ... 957
hook_menu_check_flag_value/3 ... 673, 683
hook_menu_default_option/3 ... 674, 684
hook_menu_flag_help/3 ... 673, 684
hook_menu_flag_values/3 ... 673, 683
hostname_address/2 ... 476
html_expansion/2 ... 711
html_protect/1 ... 710
html_report_error/1 ... 706
html_template/3 ... 704
html2terms/2 ... 703
http_lines/3 ... 710

I

icon_address/2 ... 710
if/3 ... 100
imports_meta_pred/3 ... 461
in/1 ... 771
in/2 ... 771
in_circle_oc/3 ... 959
in_circle_oo/3 ... 959
in_noblock/1 ... 771
in_stream/2 ... 772
inc_indentation/2 ... 1056
inccounter/2 ... 785
include/1 ... 44
incr_num_local_backtr/0 ... 569
indentation_list/2 ... 1033
inform_user/1 ... 199
inherited/1 ... 621
initial/0 ... 550
initialize_db/0 ... 727
insert/3 ... 809, 920
insert_after/3 ... 921
insert_begin/3 ... 921
insert_comments_in_beginning/3 ... 1054
insert_end/3 ... 921
insert_last/3 ... 291
insert_parsed/3 ... 1060
insert_top/3 ... 920
inside_proto/1 ... 1058
instance_codes/2 ... 637
instance_of/2 ... 636
intercept/3 ... 177
interface/2 ... 637
interp_file/2 ... 858
intersect_vars/3 ... 447
intersection/3 ... 292
intset_delete/3 ... 292
intset_in/2 ... 292
intset_insert/3 ... 292
intset_sequence/3 ... 292
is/2 ... 151
is_array/1 ... 775
is_assoc/1 ... 778
is_connected_to_java/0 ... 765
is_dictionaries/1 ... 1009
issue_debug_messages/1 ... 410

J

java_add_listener/3 ... 756
java_connect/2 ... 753
java_create_object/2 ... 754
java_debug/1 ... 765
java_debug_redo/1 ... 765
java_delete_object/1 ... 755
java_disconnect/0 ... 753
java_get_value/2 ... 756
java_invoke_method/2 ... 755
java_query/2 ... 764
java_remove_listener/3 ... 757
java_response/2 ... 764
java_set_value/2 ... 756
java_start/0 ... 752
java_start/1 ... 752
java_start/2 ... 753
java_stop/0 ... 753
java_use_module/1 ... 754
join_socket_interface/0 ... 764
just_benchmarks/0 ... 389

K

keysort/2 ... 296
keyword/1 ... 730, 737

L

labeling/1 ... 983
last/2 ... 291
length/2 ... 288, 922
length_next/2 ... 923
length_prev/2 ... 923
letter_match/2 ... 792
library_directory/1 ... 66, 167
linda_client/1 ... 771
linda_timeout/2 ... 772
line_count/2 ... 162
line_position/2 ... 163
linker_and_opts/2 ... 667
list_concat/2 ... 291
list_insert/2 ... 291, 787
list_lookup/3 ... 292
list_lookup/4 ... 292
list_to_assoc/2 ... 780
list_to_conj/2 ... 435
list_to_conj/3 ... 435
list_to_disj/2 ... 436
list_to_disj2/2 ... 439
list_to_list_of_lists/2 ... 294
llist_to_conj/2 ... 437
llist_to_disj/2 ... 438
lock_atom/1 ... 468
lock_file/3 ... 433
look_ahead/3 ... 1062
look_first_parsed/2 ... 1061
lookup_check_field/6 ... 1042
lookup_check_interface_fieldValue/8 ... 1043
lookup_check_node/4 ... 1042
lookup_field/4 ... 1043
lookup_field_access/4 ... 1044
lookup_fieldTypeId/1 ... 1044
lookup_get_fieldType/4 ... 1044
lookup_route/5 ... 1044
lookup_set_def/3 ... 1045
lookup_set_extern_prototype/4 ... 1046
lookup_set_prototype/4 ... 1045
ls/2 ... 495
ls/3 ... 495
lub/2 ... 985

M

main/1 ... 388
make/1 ... 487
make_actmod/2 ... 46
make_directory/1 ... 309
make_directory/2 ... 309
make_dirpath/1 ... 309
make_dirpath/2 ... 309
make_exec/2 ... 44
make_option/1 ... 488
make_persistent/2 ... 728
make_po/1 ... 45, 297
make_wam/1 ... 297
map/3 ... 517, 781
map/4 ... 518
map_assoc/2 ... 781
map_assoc/3 ... 781
match_pattern/2 ... 791
match_pattern/3 ... 791
match_pattern_pred/2 ... 792
match_posix/2 ... 1070, 1073
match_posix/4 ... 1070, 1074
match_posix_matches/3 ... 1070, 1074
match_posix_rest/3 ... 1070, 1074
match_pred/2 ... 1071, 1075
match_shell/2 ... 1070, 1073
match_shell/3 ... 1069, 1073

match_struct/4 ..... 1071, 1075
max_assoc/3 778
member_0/2 787
member_var/2 447
memberchk/2 787
menu/1 675
menu/2 675
menu/3 675
menu/4 675
menu_default/3 ..... 673, 682
menu_opt/6 ..... 673, 682
merge/3 812
merge_tree/2 1011
message/1 199
message/2 197
message_lns/4 198
messages/1 198
mfclause/2 242
mfstringValue/5 1017
mfstringValue/7 1020
min_assoc/3 778
minimum/3 518
mkdir_perm/2 499
mktemp/2 314
mktemp_in_tmp/2 314
mode_of_module/2 298
modif_time/2 316
modif_time0/2 316
module_address/2 590
module_of/2 298
more_solutions/1 566
most_general_instance/3 445
most_specific_generalization/3 445
move_execution_top/1 566
move_file/2 494
move_files/2 493
multibarchart/10 902
multibarchart/8 902
multifile/1 47
my_url/1 708

N

name/2 141
name_value/2 489
neighbors/3 797
neq/3 681
new/2 635
new_array/1 775
new_atom/1 325
new_interp/1 ..... 697, 857
new_interp/2 697
new_interp_file/2 698
new_measure/0 569
newer/2 490
next/2 920
next_on_circle/2 958
nl/0 172
nl/1 172
no_path_file_name/2 423
no_swapslash/3 320
no_tr_nl/2 498
nocontainsx/2 291
node_id/2 969
nodeDeclaration/4 ..... 1021, 1047
nofileerrors/0 185
nogc/0 185
nonsingle/1 283
normal_message/2 490
not_empty/3 916
not_empty/4 915
not_in_circle_oc/3 958
not_measure/0 569
note/1 199
note_message/1 407
note_message/2 407
note_message/3 407
nth/3 289
null_ddlist/1 919
number_agents/1 560
number_chars/2 260
number_codes/2 143
numbervars/3 253

O

once/1 265
op/3 255
open/3 ..... 159, 427
open/4 160
open_client/2 772
open_DEF/5 1031
open_EXTERNPROTO/5 1031
open_input/2 397
open_node/6 1029
open_null_stream/1 397
open_output/2 397
open_predicate/1 191
open_PROTO/4 1030
open_Script/5 1032

optional_message/2 408
optional_message/3 408
ord_delete/3 809
ord_disjoint/2 812
ord_intersect/2 810
ord_intersection/3 810
ord_intersection_diff/4 810
ord_list_to_assoc/2 781
ord_member/2 809
ord_subset/2 811
ord_subset_diff/3 811
ord_subtract/3 810
ord_test_member/3 810
ord_union/3 811
ord_union_change/3 812
ord_union_diff/4 811
ord_union_symdiff/4 812
otherwise/0 102
out/1 ..... 771, 1039
out/3 1039
out_stream/2 772
output_error/1 1013
output_html/1 703
output_to_file/2 431

P

parse_term/3 414
parse_term/4 415
parser/2 1047
passerta_fact/1 726
passertz_fact/1 726
pattern/2 863
pause/1 301
peek_byte/1 261
peek_byte/2 261
peek_char/1 263
peek_char/2 263
peek_code/1 171
peek_code/2 170
percentbarchart1/7 867
percentbarchart2/7 873
percentbarchart3/7 876
percentbarchart4/7 880
performance/3 1083
persistent_dir/2 ..... 681, 729, 736
persistent_dir/4 ..... 682, 729
phrase/2 275
phrase/3 275
pipe/2 165
pitm/2 983
pl2sqlInsert/2 745
point_to/3 799
pop_active_config/0 491
pop_ciao_flag/1 184
pop_global/2 537
pop_name_value/1 491
pop_prolog_flag/1 183
popen/3 311
portray/1 254
portray_attribute/2 254
portray_clause/1 253
portray_clause/2 252
postgres2sqltype/2 741
postgres2sqltypes_list/2 741
powerset/2 294
predicate_property/2 327
predicate_property/3 327
pretract_fact/1 726
pretractall_fact/1 726
pretty_print/2 451
pretty_print/3 451
pretty_print/4 451
prettyvars/1 253
prettyvars/2 815
prev/2 920
print/1 252
print/2 252
print_measures/0 569
printable_char/1 253
prolog_flag/3 182
prolog_query/2 764
prolog_response/2 765
prolog_server/0 759
prolog_server/1 760
prolog_server/2 760
prompt/2 184
prune_dict/3 814
push_active_config/1 491
push_ciao_flag/2 184
push_dictionaries/3 1057
push_global/2 537
push_goal/1 561
push_goal/3 560
push_name_value/3 491
push_prolog_flag/2 183
push_whitespace/3 1057
put_assoc/4 782
put_assoc/5 782
put_byte/1 262

put_byte/2 ... 262
put_char/1 ... 263
put_char/2 ... 263
put_code/1 ... 172
put_code/2 ... 171

Q

q_delete/3 ... 805
q_empty/1 ... 805
q_insert/3 ... 805
q_member/2 ... 805
query_requests/2 ... 760
query_solutions/2 ... 760

R

random/1 ... 807
random/3 ... 807
random_color/1 ... 863
random_darkcolor/1 ... 864
random_lightcolor/1 ... 863
random_pattern/1 ... 864
rd/1 ... 772
rd/2 ... 772
rd_findall/3 ... 772
rd_noblock/1 ... 772
reachability/4 ... 825
read/1 ... 245
read/2 ... 245
read_event/1 ... 566
read_page/2 ... 1004
read_pr/2 ... 979
read_term/2 ... 246
read_term/3 ... 246
read_terms_file/2 ... 1040
read_top_level/3 ... 246
read_vrml_file/2 ... 1040
readf/2 ... 497
reading/4 ... 1023
reading/5 ... 1025
reading/6 ... 1029
rebuild_foreign_interface/1 ... 669
rebuild_foreign_interface_explicit_decls/2 ... 670
rebuild_foreign_interface_object/1 ... 671
receive_confirm/2 ... 699
receive_event/2 ... 699
receive_list/2 ... 699
receive_result/2 ... 698
recorda/3 ... 331
recorded/3 ... 332
recordz/3 ... 331
recycle_term/2 ... 449
reduce_indentation/3 ... 1057
register_module/1 ... 490
release/1 ... 567
release_all_for_unwinding/0 ... 567
release_remote/1 ... 567
release_some_suspended_thread/0 ... 567
remove_all_elements/3 ... 922
remove_code/3 ... 1062
remove_comments/4 ... 1034
remove_menu_config/1 ... 677
rename/2 ... 815
rename_file/2 ... 318
repeat/0 ... 101
replace_all/4 ... 1071, 1075
replace_characters/4 ... 320
replace_first/4 ... 1071, 1075
replace_strings/3 ... 498
replace_strings_in_file/3 ... 498
reserved_words/1 ... 1006
restore_menu_config/1 ... 677
restore_menu_flags/2 ... 678
restore_menu_flags_list/1 ... 678
retract/1 ... 241
retract_fact/1 ... 189, 727, 736
retract_fact_nb/1 ... 190
retractall/1 ... 241
retractall_fact/1 ... 190, 727
retrieve_goal/2 ... 562
retrieve_list_of_values/2 ... 986
retrieve_range/2 ... 984
retrieve_store/2 ... 984
reverse/2 ... 286
reverse/3 ... 286
reverse_parsed/2 ... 1061
rewind/2 ... 922
rooted_subgraph/3 ... 799
run_tester/10 ... 1077
running_queries/2 ... 760

S

safe_write/2 ... 479
save_addr_actmod/1 ... 590
save_end_execution/1 ... 566
save_init_execution/1 ... 566
save_menu_config/1 ... 676
scattergraph_b1/12 ... 887
scattergraph_b1/8 ... 887
scattergraph_b2/12 ... 897
scattergraph_b2/8 ... 896
scattergraph_w1/12 ... 889
scattergraph_w1/8 ... 888
scattergraph_w2/12 ... 898
scattergraph_w2/8 ... 898
second_prompt/2 ... 246
see/1 ... 329
seeing/1 ... 329
seen/0 ... 329
select/3 ... 288
select_socket/5 ... 475
self/1 ... 590, 621
send_event/1 ... 565
send_info_to_developers/0 ... 389
send_signal/1 ... 178
send_silent_signal/1 ... 178
send_term/2 ... 699
sequence_to_list/2 ... 294
serve_socket/3 ... 479
set_ciao_flag/2 ... 184
set_cookie/2 ... 707
set_debug_mode/1 ... 46, 298
set_debug_module/1 ... 298
set_debug_module_source/1 ... 298
set_env/2 ... 304
set_environment/3 ... 1060
set_exec_mode/2 ... 317
set_exec_perms/2 ... 499
set_fact/1 ... 191
set_general_options/1 ... 989
set_global/2 ... 537
set_goal_cancelled/1 ... 565
set_goal_det/1 ... 562
set_goal_failed/1 ... 565
set_goal_finished/1 ... 563
set_goal_nondet/1 ... 562
set_goal_not_executed/1 ... 563
set_goal_rem_executing/1 ... 563
set_goal_tobacktrack/1 ... 564
set_goal_toreexecute/1 ... 564
set_input/1 ... 161
set_menu_flag/3 ... 676
set_name_value/2 ... 489
set_nodebug_mode/1 ... 46, 298
set_nodebug_module/1 ... 298
set_output/1 ... 161
set_owner/2 ... 497
set_parsed/3 ... 1060
set_perms/2 ... 499
set_prolog_flag/1 ... 186
set_prolog_flag/2 ... 182
set_stream/3 ... 411
setarg/3 ... 577
setcounter/2 ... 785
setenvstr/2 ... 304
setof/3 ... 233, 471
setproduct/3 ... 812
sformat/3 ... 278
shell/0 ... 310
shell/1 ... 310
shell/2 ... 310
shell_s/0 ... 760
show_handler/1 ... 565
show_menu_config/1 ... 677
show_menu_configs/0 ... 677
show_message/2 ... 409
show_message/3 ... 409
show_message/4 ... 409
simple_message/1 ... 407
simple_message/2 ... 407
skip_code/1 ... 171
skip_code/2 ... 171
skip_line/0 ... 171
skip_line/1 ... 171
socket_accept/2 ... 474
socket_recv/2 ... 476
socket_recv_code/3 ... 475
socket_send/2 ... 475
socket_shutdown/2 ... 476
sort/2 ... 295
sort_dict/2 ... 814
space/1 ... 676
split/4 ... 519
sql__attribute/4 ... 745
sql__relation/3 ... 745
sql_goal_tr/2 ... 743
sql_persistent_tr/2 ... 743
srandom/1 ... 808
standard_ops/0 ... 257
start_socket_interface/2 ... 763
start_thread/1 ... 560

start_threads/0 765
start_vrmlScene/4 1033
statistics/0 323
statistics/2 323
stop_parse/2 1061
stop_socket_interface/0 764
stream_code/2 164
stream_property/2 267
stream_to_string/2 430
stream_to_string/3 431
string/3 403
string_to_file/2 430
string2term/2 414
strip_clean/2 1059
strip_exposed/2 1059
strip_from_list/2 1058
strip_from_term/2 1059
strip_interface/2 1059
strip_restricted/2 1059
sub_atom/4 149
sub_atom/5 265
sub_times/3 1086
subtract/3 788
sum_list/2 789
sum_list/3 790
sum_list_of_lists/2 790
sum_list_of_lists/3 790
suspend/0 567
sybase2sqltype/2 741
sybase2sqltypes_list/2 740
symbolic_link/2 494
symbolic_link/3 494
SYSCALL/1 516
system/1 311
system/2 311
system_error_report/1 320
system_lib/1 459

T

tab/1 173
tab/2 173
tablewidget1/4 905
tablewidget1/5 905
tablewidget2/4 909
tablewidget2/5 909
tablewidget3/4 911
tablewidget3/5 911
tablewidget4/4 913
tablewidget4/5 913
tcl_delete/1 693
tcl_eval/3 692
tcl_event/3 693
tcl_new/1 692
tcltk/2 698
tcltk_raw_code/2 ..... 698, 857
tell/1 329
telling/1 329
term_size/2 441
terms_file_to_vrml/2 1002
terms_file_to_vrml_file/2 1003
terms_to_vrml/2 1003
terms_to_vrml_file/2 1003
this_module/1 207
throw/1 178
time/1 301
tk_event_loop/1 694
tk_main_loop/1 694
tk_new/2 694
tk_next_event/2 695
to_list/2 920
token_read/3 1065
tokeniser/2 1065
told/0 329
top/2 922
topd/0 685
touch/1 499
transpose/2 ..... 396, 799
true/0 101
true/1 347
trust/1 346
try_finally/3 497
ttydisplay/1 334
ttydisplay_string/1 334
ttydisplayq/1 334
ttyflush/0 334
ttyget/1 333
ttyget1/1 333
ttynl/0 333
ttyput/1 333
ttyskip/1 333
ttyskipeol/0 334
ttytab/1 333
type_compatible/2 740
type_union/3 740

U

ugraph2term/2 ... 686
umask/2 ... 308
uncycle_term/2 ... 449
undo/1 ... 577
undo_force_lazy/1 ... 46
unfold_tree/2 ... 1090
unfold_tree_dic/3 ... 1091
uni_type/2 ... 681
unify_with_occurs_check/2 ... 266
union/3 ... 293
union_idlists/3 ... 788
unload/1 ... 45, 298
unlock_atom/1 ... 468
unlock_file/2 ... 433
unregister_module/1 ... 490
update/0 ... 458
update_assoc/5 ... 783
update_attribute/2 ... 201
update_files/0 ... 728
update_files/1 ... 728
url_info/2 ... 708
url_info_relative/3 ... 709
url_query/2 ... 707
url_query_amp/2 ... 707
url_query_values/2 ... 708
use_class/1 ... 638
use_module/1 ... 43, 297
use_module/2 ... 44, 297
use_module/3 ... 297
use_package/1 ... 45
using_windows/0 ... 318

V

valid_attributes/2 ... 917
valid_format/4 ... 916
valid_table/2 ... 917
valid_vectors/4 ... 917
variant/2 ... 445
varnamesl2dict/2 ... 815
vars_names_dict/3 ... 815
varsbag/3 ... 447
varset/2 ... 447
varset_in_args/2 ... 447
vectors_format/4 ... 916
verbose_message/1 ... 488
verbose_message/2 ... 488
verify_attribute/2 ... 202
vertices/2 ... 798
vertices_edges_to_lgraph/3 ... 803
vertices_edges_to_ugraph/3 ... 797
vertices_edges_to_wgraph/3 ... 801
vmember/2 ... 681
vpath/1 ... 490
vpath_mode/3 ... 490
vrml_file_to_terms/2 ... 1001
vrml_file_to_terms_file/2 ... 1002
vrml_http_access/2 ... 1004
vrml_in_out/2 ... 1004
vrml_to_terms/2 ... 1003
vrml_web_to_terms/2 ... 1001
vrml_web_to_terms_file/2 ... 1002

W

wait/3 ... 313
waiting/1 ... 566
warning/1 ... 198
warning_message/1 ... 406
warning_message/2 ... 406
warning_message/3 ... 406
wellformed_body/3 ... 244
when/2 ... 582
where/1 ... 458
whitespace/2 ... 402
whitespace0/2 ... 402
winpath/2 ... 318
winpath/3 ... 319
winpath_c/3 ... 319
working_directory/2 ... 309
write/1 ... 251
write/2 ... 250
write_assertion/6 ... 455
write_assertion/7 ... 455
write_assertion_as_comment/6 ... 455
write_assertion_as_comment/7 ... 456
write_assertion_as_double_comment/6 ... 456
write_assertion_as_double_comment/7 ... 456
write_attribute/1 ... 253
write_canonical/1 ... 252
write_canonical/2 ... 251
write_list1/1 ... 252
write_pr/2 ... 979
write_string/1 ... 402
write_string/2 ... 402
write_term/2 ... 249
write_term/3 ... 249
write_terms_file/2 ... 1040
write_vrml_file/2 ... 1040

writef/2 498
writef/3 498
writef_list/2 499
writef_list/3 498
writeq/1 251
writeq/2 251

X

xdr_tree/1 1089
xdr_tree/3 1089
xdr_xpath/2 1091
xdr2html/2 1090
xdr2html/4 1090
xml_index/1 1095
xml_index_query/3 1095
xml_index_to_file/2 1095
xml_parse/3 1094
xml_parse_match/3 1094
xml_query/3 1096
xml_search/3 1094
xml_search_match/3 1095
xml2terms/2 704

Property Index

=

=/2 129
==/2 135

A

assert_body_type/1 438
atom/1 122
atomic/1 125

B

bind_ins/1 119

C

class_name/1 638
class_source/1 639
clique/1 359
clique_1/1 359
compat/2 116
const_head/1 133
constraint/1 360
constructor/1 638
covered/1 360
covered/2 360

D

davinci_command/1 686
deprecated/1 117
do_not_free/2 663
docstring/1 354

E

equiv/2 118
error_free/1 119
eval/1 118
exception/1 360
exception/2 360
expander_pred/1 462

F

fails/1 361
field_Id/1 1048
filter/2 119
finite_solutions/1 361
float/1 124
foreign/1 662
foreign/2 662
foreign_low/1 662
foreign_low/2 662
fuzzybody/1 608

G

ground/1 126

H

have_choicepoints/1 361
head_pattern/1 350

I

indep/1 361
indep/2 361
inst/2 116
instance/2 ..... 369, 445
instance_id/1 639
integer/1 123
interface_name/1 639
interface_source/1 639
is_det/1 361
iso/1 116

L

lgraph/1 687
linear/1 362
list1/2 290

M

member/2 113
memo/1 119
method_spec/1 639
mshare/1 362
mut_exclusive/1 362

N

nabody/1 ... 352
native/1 ... 118
native/2 ... 118
needs_state/1 ... 663
no_choicepoints/1 ... 362
no_exception/1 ... 363
no_exception/2 ... 363
no_rtcheck/1 ... 118
no_signal/1 ... 363
no_signal/2 ... 363
non_det/1 ... 363
nonground/1 ... 363
nonvar/1 ... 122
not_covered/1 ... 363
not_fails/1 ... 364
not_further_inst/2 ... 117
not_mut_exclusive/1 ... 364
num_solutions/2 ... 364
number/1 ... 125

P

parse/1 ... 1018
pe_type/1 ... 119
possibly_fails/1 ... 365
possibly_nondet/1 ... 365

R

regtype/1 ... 118
relations/2 ... 365
returns/2 ... 662

S

sideff/2 ... 117
sideff_hard/1 ... 365
sideff_pure/1 ... 365
sideff_soft/1 ... 366
signal/1 ... 366
signal/2 ... 366
signals/2 ... 366
size/2 ... 366
size/3 ... 366
size_lb/2 ... 366
size_metric/3 ... 367
size_metric/4 ... 367
size_o/2 ... 366
size_of/3 ... 662
size_ub/2 ... 367
solutions/2 ... 364
sourcenames/1 ... 47
steps/2 ... 367
steps_lb/2 ... 367
steps_o/2 ... 367
steps_ub/2 ... 368
sublist/2 ... 293
subordlist/2 ... 293

T

tau/1 ... 368
terminates/1 ... 368
test_type/2 ... 368
throws/2 ... 368
ttr/3 ... 663
type/2 ... 127

U

ugraph/1 ... 687
user_output/2 ... 369

V

var/1 ... 121
virtual_method_spec/1 ... 639

W

write_option/1 ... 250

Regular Type Index

A

address/1 662
any_term/1 662
apropos_spec/1 460
argspec/1 513
arithexpression/1 156
assrt_body/1 349
assrt_status/1 353
assrt_type/1 354
atm/1 108
atm_or_atm_list/1 115
atm_or_int/1 167
atom_or_str/1 425
attributes/1 890
axis_limit/1 868

B

benchmark_usage/1 388
bltwish_interp/1 857
body/1 452
bound/1 1035
bound_double/1 1035
byte/1 661
byte_list/1 661

C

c_assrt_body/1 352
callable/1 110
canonic_html_term/1 715
canonic_xml_item/1 1096
canonic_xml_query/1 1096
canonic_xml_subquery/1 1097
canonic_xml_term/1 ..... 716, 1096
cell_value/1 907
character_code/1 114
clause/1 452
clauses/1 452
clockfreq_option/1 326
clockfreq_result/1 325
clterm/1 452
color/1 861
comparator/1 138
complex_arg_property/1 350
complex_goal_property/1 351
conj_disj_type/1 438
constant/1 110

D

datime_struct/1 304
ddlist/1 923
detcond/1 540
dgraph/1 793
dht_rpr_node_id/1 963
dictionary/1 ..... 352, 399, 1035
directoryname/1 728
dlgraph/1 793
do_options/1 500
double_list/1 661

E

elisp_string/1 769
environment/1 1036

F

faggregator/1 608
fd_item/1 982
fd_range/1 982
fd_store/1 983
fd_store_entity/1 983
fd_subrange/1 982
flag/1 452
flag_values/1 119
flt/1 107
footer/1 869
form_assignment/1 719
form_dict/1 719
form_value/1 719
format_control/1 279

G

g_assrt_body/1 353
garbage_collection_option/1 326
gc_result/1 326
gnd/1 109
gndstr/1 109

H

handler_type/1 860
header/1 869
hms_time/1 721
html_term/1 717
http_date/1 721
http_request_param/1 720

http_response_param/1 720

I

image/1 906
indexspecs/1 512
int/1 106
int_list/1 661
internal_module_id/1 208
intexpression/1 158
intlist/1 789
io_mode/1 167

J

java_constructor/1 754
java_event/1 754
java_field/1 754
java_method/1 755
java_object/1 754

K

keylist/1 296
keypair/1 296

L

lgraph/2 803
line/1 401
list/1 111
list/2 112
list_functor/1 134
list_of_lists/1 294

M

machine_name/1 753
memory_option/1 326
memory_result/1 326
menu_flag_values/1 681
message_info/1 199
message_t/1 410
message_type/1 200
meta_predname/1 728
metaspec/1 94
modulename/1 94
month/1 721
multibar_attribute/1 903

N

nlist/2 112
nnegint/1 106
non_empty_dictionary/1 400
non_empty_list/1 132
null/1 661
null_dict/1 813
num/1 107
num_code/1 115
numlist/1 789

O

old_or_new/1 400
open_option_list/1 160
operator_specifier/1 110

P

pair/1 795
parse/1 1036
path/1 541
pattern/1 ..... 500, 792, 863
popen_mode/1 311
posix_regexp/1 1075
postgrestype/1 741
predfunctor/1 354
predname/1 115
prolog_goal/1 754
property_conjunction/1 351
property_starterm/1 351
propfunctor/1 354

R

read_option/1 246
reference/1 192
row/1 906

S

s_assrt_body/1 352
sequence/2 113
sequence_or_list/2 113
shell_regexp/1 1075
shutdown_type/1 477
size/1 891
smooth/1 890
socket_type/1 477
sourcename/1 165
spec/1 393
sqltype/1 739
stream/1 166
stream_alias/1 166
string/1 114
struct/1 108
struct_regexp/1 1076
sybasetype/1 740
symbol/1 890
symbol_option/1 326
symbol_result/1 326

T

t_conj/1 438
t_disj/1 439
table/1 906
tag_attrib/1 1096
target/1 487
tclCommand/1 694
tclInterpreter/1 693
term/1 105
tick_option/1 327
tick_result/1 325
time_option/1 327
time_result/1 326
title/1 869
translation_predname/1 195
tree/1 1036
triple/1 795

U

ugraph/1 799
url_term/1 720

V

valid_base/1 149
value_dict/1 720
varnamedict/1 815
vector/1 889

W

wakeup_exp/1 582
weekday/1 721
whitespace/1 1036

X

xbarelement1/1 869
xbarelement2/1 873
xbarelement3/1 877
xbarelement4/1 881
xdr_node/1 1090
xelement/1 904

Y

yelement/1 867

Declaration Index

A

add_clause_trans/2 195
add_goal_trans/2 195
add_sentence_trans/2 194
add_term_trans/2 194
aggr/1 606
argnames/1 521

B

block/1 845

C

calls/1 341
calls/2 341
comment/2 346
comp/1 343
comp/2 343
concurrent/1 ..... 192, 619

D

data/1 ..... 192, 618
decl/1 345
decl/2 345
determinate/2 539
discontiguous/1 103
doc/2 345
dynamic/1 619

E

ensure_loaded/1 97
entry/1 344
exit/1 344
exit/2 344
export/1 ..... 92, 618, 997
extra_compiler_opts/1 664
extra_compiler_opts/2 664
extra_linker_opts/1 664
extra_linker_opts/2 665

F

facts/2 737
foreign_inline/2 665

I

impl_defined/1 103
implements/1 620
import/2 93
include/1 97
index/1 512
inherit_class/1 619
inheritable/1 618
initialization/1 104
instance_of/2 629

L

load_compilation_module/1 194

M

meta_predicate/1 94
modedef/1 345
module/2 92
module/3 91
multifile/1 103

N

new/2 630
new_declaration/1 193
new_declaration/2 193

O

on_abort/1 104
op/3 193

P

package/1 92
persistent/2 729
pred/1 340
pred/2 341
prop/1 343
prop/2 344
protocol/1 589
public/1 618

R

redefining/1 104
reexport/1 94
reexport/2 93
regtype/1 358
regtype/2 ..... 358, 547

S

success/1 342
success/2 342

T

test/1 342
test/2 342
texec/1 341
texec/2 341

U

use_active_module/2 587
use_class/1 629
use_compiler/1 664
use_compiler/2 664
use_foreign_library/1 663
use_foreign_library/2 663
use_foreign_source/1 663
use_foreign_source/2 663
use_linker/1 665
use_linker/2 665
use_module/1 93
use_module/2 93
use_package/1 97

V

virtual/1 620

Concept Index

&

&-Prolog ... 10

.

.ciaorc ... 24, 29

A

abort ... 56
abstract methods ... 620
acceptable modes ... 350
acknowledgments ... 9
active module ... 37, 585
active object ... 585
addmodule and pred(N) meta-arguments ... 617
ancestors ... 55
answer variable ... 42
assertion body syntax ... 349, 352, 353
assertions ... 67
attribute ... 618
attributed variables ... 201
Austrian Research Institute for AI ... 10
auto-documenter command args, setting ... 78
auto-documenter command, setting ... 78
auto-documenter default format, setting ... 77
auto-documenter lib path, setting ... 78
auto-documenter working dir, setting ... 78
auto-fill ... 67
auto-indentation ... 67

B

binary directory ... 1103
box-type debugger ... 49
breakpoints ... 72
Bristol University ... 10
bugs, reporting ... 1116

C

calls assertion ... 341
certainty factor ... 997
CGI ... 701
CGI executables ... 63
change, author ... 75
change, comment ... 75
changelog ... 67
changing the executables used ... 78
check assertion ... 346
Ciao basic builtins ... 8, 89
Ciao engine ... 10
Ciao mode version ... 79
Ciao preprocessor ... 10, 67, 73
Ciao top-level ... 67
ciao, global description ... 3
Ciao, why this name ... 5
ciao-users ... 1115
client installation ... 1113
CLIP group ... 9
closed ... 190
coloring, syntax ... 67
command ... 56
comment assertion ... 346
comments, machine readable ... 339
comp assertion ... 343
compatibility properties ... 355
compiler, standalone ... 33
compiling ... 69, 70
compiling programs ... 22, 23, 28, 29
compiling, from command line ... 33
compiling, Win32 ... 1107
concurrency ... 465
concurrent attribute ... 619
concurrent predicate ... 187
concurrent predicates ... 187
configuration file ... 831
constructor ... 621
contributed libraries ... 9, 843
creating executables ... 69
creep ... 54
csh-compatible shell ... 21, 1101, 1104
current input stream ... 161
current output stream ... 161
customize ... 68, 78
Cygnus Win32 ... 1107

D

data declaration ... 187
data predicate ... 187
database initialization ... 727
debug (interpreted) mode ... 49
debug options ... 54
debugger ... 49
debugging ... 55, 72
debugging, source-level ... 67, 72
decl assertion ... 345
declarations, user defined ... 103
DECsystem-10 Prolog User's Manual ... 10

depth first iterative deepening . . . 595
depth limit . . . 595
destructor . . . 622
determinate goal . . . 539
development environment . . . 24, 29, 1102, 1105, 1111
display . . . 55
Distributed Programming Model . . . 747
downloading emacs . . . 1105, 1111
downloading, latest versions . . . 1115

E

emacs interface . . . 7, 31
emacs lisp . . . 767
emacs mode . . . 67
emacs mode, loading several . . . 79
emacs mode, setting up, Win32 . . . 1112
emacs server . . . 767
emacs, download . . . 1105, 1111
emacs, intro . . . 24, 29
engine directory . . . 8, 89
engine module . . . 509
entry assertion . . . 344
environment variable definitions . . . 1101
environment variables . . . 87
environment variables, setup . . . 21
executable . . . 33
executables, compressed . . . 37
executables, dynamic . . . 35
executables, generating . . . 23, 28
executables, how to run . . . 34
executables, lazy load . . . 36
executables, self-contained . . . 36
executables, static . . . 36
executables, types . . . 35
exit assertion . . . 344
extensibility . . . 4

F

fail . . . 55
false assertion . . . 347
feature terms . . . 521
formatting commands . . . 339
formatting conventions, for emacs . . . 67
Function applications . . . 525
Functional definitions . . . 526

G

German Puebla . . . 9

H

hard side-effects . . . 365
help . . . 24, 27, 29, 56, 1112
help, unix . . . 22
help, windows . . . 28
HTML . . . 701
HTTP . . . 701

I

IMDEA Software Institute . . . 9
INFOPATH . . . 1104
inheritable interface . . . 618
inheritance relationship . . . 620
initialization clauses . . . 619
initialization file . . . 24, 29
INRIA . . . 10
installation, checking the . . . 1105
installation, Mac OS X, full instructions . . . 1102
installation, Mac OS X, summary . . . 1101
installation, network based . . . 1103
installation, Un*x, full instructions . . . 1102
installation, Un*x, summary . . . 1101
installation, Windows clients . . . 1113
installation, Windows server . . . 1113
installation, Windows, from binaries . . . 1111
installation, Windows, from sources . . . 1107
instantiation properties . . . 355
interface inheritance . . . 620
interfaces . . . 617
interpreting . . . 69, 70
iso . . . 8, 229
ISO-Prolog . . . 4, 157
ISO-Prolog builtins . . . 8, 229
iso-prolog, compliance . . . 4
iterative-deepening . . . 595

J

Java event handling from Prolog . . . 750
Java exception handling from Prolog . . . 752
Java to Prolog interface . . . 759
Johan Andersson . . . 87
Johan Bevemyr . . . 87

K

K.U. Leuven ... 10
key sequences ... 68
keyboard ... 6

L

leap ... 54
lib library ... 8, 89
library directory ... 1103
limitations, architecture-specific ... 1115
Linkoping U. ... 10
loading mode ... 50
loading programs ... 22, 28, 69
locating errors ... 72
LogIn ... 42
LPdoc ... 3
lpdoc command args, setting ... 78
lpdoc command, setting ... 78
lpdoc default format, setting ... 77
lpdoc lib path, setting ... 78
lpdoc working dir, setting ... 78
lpmake ... 831
lpmake autodocumentation ... 831

M

mailing list ... 1115
main module ... 70
make ... 831
MANPATH ... 1104
manual, printing ... 24, 27, 29, 1112
manual, tour ... 7
manuals, printing ... 24, 29
Maria Jose Garcia de la Banda ... 9
Masanobu Umeda ... 87
Mats Carlsson ... 87
MCC ... 10
Melbourne U. ... 10
modular interface ... 37
module qualification ... 91
modules, active ... 37
Monash U. ... 10
moving changelog entries ... 76
multi-evaluated ... 618
multiarchitecture support ... 1106

N

Naming term arguments ... 521
New Mexico State University ... 10
nodebug ... 55
nospy ... 56
notation ... 5

O

Other functionality ... 527
overriden ... 619

P

parallel Prolog ... 10
parallelizing compiler ... 10
parametric type functor ... 358
PATH ... 1103
path alias ... 165
patterns ... 791, 1069
Pedro Lopez ... 9
Peter Olin ... 87
PiLLoW on-line tutorial ... 701
Platform independence ... 747
Polymorphism ... 611
pred assertion ... 340, 341
Predefined evaluable functors ... 525
preprocessing programs ... 73
preprocessor command args, setting ... 78
preprocessor command, setting ... 78
print ... 55
printdepth ... 56
printing, manual ... 24, 27, 29, 1112
program development environment ... 67
program transformations ... 67
programming environment ... 7, 31
prolog flag ... 181, 197
Prolog server ... 759, 760
Prolog shell scripts ... 63
Prolog to Java Interface Structure ... 749
Prolog to Java Interface Structure. Java side ... 749
Prolog to Java Interface Structure. Prolog side ... 749
prolog-emacs interface ... 767
prop assertion ... 343, 344
properties of computations ... 355
properties of execution states ... 355
properties, basic ... 105
properties, native ... 359
protected ... 618
public domain ... 1

public interface 618
pure Prolog 509

Q

query 41
Quoting functors 526

R

records 8, 507, 521
recursive level 42
references, to Ciao 5
referring to Ciao 5
regtype assertion 358
regular expressions 791
regular type expression 358
reporting bugs 1116
retry 55
running programs 22, 23, 28, 29
running unit tests 379

S

script header, inserting automatically 72
scripts 23, 29, 1103
sh-compatible shell 22, 1102, 1104
sharing sets 362
shortcut, windows 1111
SICS 10, 87
SICStus Prolog 10
skip 54
Socket implementation 763
soft side-effects 366
Some scoping issues 527
source directory 1102
source-level debugging 67, 72
spy 56
standard total ordering 135
static checks 67
status, this manual 3
style sheets 21, 27, 1105, 1112
subterm 56
success assertion 342
super class 619
Swedish Institute of Computer Science 10
Syntax-based highlighting 67

T

tar 1102
Technical University of Madrid 9
test assertion 342, 379
texec assertion 341
top-level shell, starting, unix 22
top-level shell, starting, windows 27
toplevel command args, setting 78
toplevel command, setting 78
tour, of the manual 7
tracing the source code 67
troubleshooting 1101, 1108, 1111
true assertion 347
trust assertion 346

U

U. of Arizona 10
unify 56
uninstalling 1102, 1105
unit tests 379
UPM 10
user module 91
user modules, debugging 49
user setup 21
users mailing list 1115

V

variables 55
version control 67
version maintenance mode for packages 75
version number 74
virtual 620

W

WAM 10
why the name Ciao 5
windows shortcut 1111
write 55
WWW, interfacing with 701

X

XML 701

Author Index

A

A. Ciepielewski 49, 59
Amadeo Casas 525, 543, 545, 993
Angel Fernandez Pineda 423, 457, 461, 611, 617, 629, 635, 641, 997
Anil Nair 511
Arsen Kostenko . . . 927, 929, 935, 937, 939, 941, 953, 957, 961, 971, 975, 979

C

Christian Holzbaur 201, 597, 601
Claudio Ochoa 987
Claudio Vaucheret 539, 595, 605

D

Daniel Cabeza . . 33, 41, 49, 59, 63, 65, 67, 91, 97, 99, 103, 105, 121, 129, 135, 151, 159, 169, 181, 187, 193, 197, 201, 205, 211, 233, 239, 245, 249, 255, 259, 265, 275, 301, 323, 335, 371, 401, 413, 421, 423, 433, 449, 473, 515, 517, 521, 525, 577, 579, 585, 593, 597, 601, 701, 703, 713, 715, 723, 807, 823, 827, 841, 1101, 1111, 1115
David H.D. Warren 233
David Trallero Mena 919, 989, 1077, 1083

E

Edison Mera 33, 49, 59, 197, 259, 359, 377, 379, 487, 517, 813, 999

F

Francisco Bueno . . . 339, 355, 359, 413, 427, 455, 479, 511, 539, 589, 605, 685, 733, 735, 787, 793, 797, 803, 813, 829

G

German Puebla 339
Goran Smedback 1001, 1005, 1007, 1009, 1013, 1015, 1017, 1019, 1021, 1023, 1035, 1039, 1041, 1047, 1049, 1063, 1065

I

Isabel Martin Garcia . . . 847, 857, 859, 861, 865, 871, 875, 879, 883, 893, 901, 905, 909, 911, 913, 915

J

Jesus Correas 747, 749, 759, 763
Jose F. Morales 205, 209, 323, 521, 525, 571, 575, 645, 661, 667, 669
Jose Manuel Gomez Perez . . 433, 723, 981, 1089, 1093

K

K. Shen 49, 59

L

Lena Flood 775, 809

M

Manuel C. Rodriguez 49, 59, 67
Manuel Carro 187, 201, 205, 245, 249, 255, 301, 323, 385, 391, 411, 433, 465, 471, 473, 517, 571, 575, 577, 579, 581, 593, 595, 645, 661, 667, 669, 777, 797, 819, 835, 837, 839, 841, 981, 1001, 1101, 1111, 1115
Manuel Hermenegildo . . . 21, 27, 63, 67, 99, 105, 121, 129, 135, 151, 233, 249, 259, 339, 349, 355, 359, 371, 373, 413, 481, 487, 493, 521, 525, 585, 593, 595, 701, 703, 723, 821, 827, 831, 1101, 1111, 1115
Mats Carlsson . . . 49, 59, 159, 169, 181, 233, 249, 323, 797
Montse Iglesias Urraca 689, 697

O

Oscar Portela Arjona 421

P

Pablo Chico 777
Pedro Lopez 355, 359

R

Remy Haemmerle 579, 595, 845
Richard A. O'Keefe 233, 249, 295, 797
Robert Manchek 835

S

Sacha Varma 703
Samir Genaim 597, 601
Sergio Guadarrama 605

T

T. Chikayama 49, 59
The CLIP Group . . . 31, 33, 41, 89, 91, 141, 177, 229, 231, 239, 259, 267, 269, 271, 277, 283, 295, 299, 331, 337, 381, 383, 399, 405, 429, 441, 445, 447, 451, 507, 509, 643, 673, 689, 723, 731, 747, 767, 773, 789, 791, 831, 843, 1001, 1069, 1099
Tom Howland 511

Global Index

This is a global index containing pointers to places where concepts, predicates, modes, properties, types, applications, etc., are referred to in the text of the document.

!

!!/0 575
!/0 ..... 99, 100

#

/2 157
##/2 605
#>/2 605

$

$/1 701
$/2 ..... 521, 673, 701
$~/3 ..... 521, 522
$cancellation/1 ..... 546, 551
$clean_measures/0 ..... 546, 559
$enter_mutex/1 ..... 546, 558
$enter_mutex_remote/1 ..... 546, 558
$enter_mutex_self/0 ..... 546, 558
$exit_mutex/1 ..... 546, 559
$exit_mutex_remote/1 ..... 546, 559
$exit_mutex_self/0 ..... 546, 559
$factsdb$cached_goal/3 ..... 733, 735, 736
$find_goal/3 ..... 546, 551
$goal_available/1 ..... 546, 551
$goal_cancelled/1 ..... 546, 555
$goal_det/1 ..... 546, 552
$goal_failed/1 ..... 546, 555
$goal_finished/1 ..... 546, 553
$goal_not_executed/1 ..... 546, 552
$goal_rem_executing/1 ..... 546, 553
$goal_tobacktrack/1 ..... 546, 554
$goal_toreexecute/1 ..... 546, 554
$incr_num_local_backtr/0 ..... 546, 560
$is_persistent/2 ..... 675, 681, 725, 729
$meta_call/1 ..... 515, 516
$more_solutions/1 ..... 546, 556
$move_execution_top/1 ..... 546, 557
$new_measure/0 ..... 546, 560
$nodebug_call/1 ..... 515, 516
$not_measure/0 ..... 546, 560
$number_agents/1 ..... 546, 550
$print_measures/0 ..... 546, 559
$push_goal/3 ..... 546, 551
$read_event/1 ..... 546, 556
$release/1 ..... 546, 557
$release_all_for_unwinding/0 ..... 546, 558
$release_remote/1 ..... 546, 557
$release_some_suspended_thread/0 ..... 546, 558
$retrieve_goal/2 ..... 546, 552
$save_end_execution/1 ..... 546, 556
$save_init_execution/1 ..... 546, 556
$send_event/1 ..... 546, 556
$set_goal_cancelled/1 ..... 546, 555
$set_goal_det/1 ..... 546, 552
$set_goal_failed/1 ..... 546, 555
$set_goal_finished/1 ..... 546, 554
$set_goal_nondet/1 ..... 546, 552
$set_goal_not_executed/1 ..... 546, 553
$set_goal_rem_executing/1 ..... 546, 553
$set_goal_tobacktrack/1 ..... 546, 554
$set_goal_toreexecute/1 ..... 546, 554
$show_handler/1 ..... 546, 555
$start_thread/1 ..... 546, 550
$suspend/0 ..... 546, 557
$waiting/1 ..... 546, 557

&

&!/1 543
&!/2 543
&!>/2 543
&&!/1 543
&&!/2 543
&&!>/2 543
&&/1 543
&&/2 543
&&>/2 543
&-Prolog 10, 87
&/1 543
&/2 543
&?/1 543
&>/2 543

,

',$predicate_property'/2 328
',$xml_search_match/3 1094
',','/2 113
',<-'/1 593
',<-'/2 593
'bf/af' 594
'bf/bfall' 593, 594
'persdb/ll' 725, 726

*

* /2 157
** /2 157
*/1 ..... 371, 546, 547, 571, 572
*/2 ..... 351, 371, 372, 546, 549, 571, 572

,

,/2 99

-

- /1 156
- /2 157
-- /1 156
--/1 ..... 493, 496
-->/2 ..... 231, 383, 546
-/1 ..... 371, 373, 496, 546, 547, 571
-/2 ..... 292, 371, 372, 373, 374, 546, 548, 571, 572
->/2 ..... 99, 100

.

. /2 527
.&./2 982
../2 982
./2 ..... 43, 45
./my_program -file input.txt -file input2.txt --output_file out.txt -create-dir --decode --unsorte 391
.=./2 ..... 597, 601, 982
.=</2 ..... 597, 601, 982
.>. 599
.>./2 ..... 597, 601, 982
.>=./2 ..... 597, 601, 982
.</2 ..... 597, 601, 982
.<>./2 ..... 597, 601, 982
.ciaorc 23, 24, 28, 29, 41
.emacs 24, 29, 67, 85, 86, 767
.tar files 827

/

/ /2 157
// /2 157
/\ /2 157
/bin/sh 311
/bin/sh.exe 35, 1107
/etc/bashrc 1102, 1105
/etc/csh.cshrc 1102, 1105
/etc/csh.login 1102, 1105
/etc/skel 1102, 1105
/usr/share/emacs/.../lisp/site-init.pl ... 1102, 1105

:

:#/2 ..... 605, 606
::/2 ..... 340, 481, 589, 590, 673
:= /2 526
:=/1 605
:=/2 605
:~/1 605
:~/2 ..... 605, 607

;

;/2 99

=

=../2 ..... 129, 132
=/2 129
=:=/2 ..... 151, 155
==/2 ..... 135, 797
=~/1 1069
=>/1 605
=>/2 ..... 340, 521, 673
=>/4 ..... 605, 608
=\=/2 ..... 151, 156
=</2 ..... 151, 153

?

? /2 525
?/1 ..... 371, 373, 546, 547, 571
?/2 ..... 371, 372, 373, 374, 546, 548, 571, 572
?=/2 539
?\=/2 539

@

@/1 ..... 371, 373, 546, 547, 571
@/2 ..... 371, 372, 373, 375, 546, 548, 571, 572
@=</2 ..... 135, 137
@>/2 ..... 135, 137
@>=/2 ..... 135, 137
@</2 ..... 135, 136

[

['-file', 'input.txt', '-file', 'input2.txt',
'--output_file', 'out.txt', '-create-dir',
'--decode', '--unsorte'] 391

|

| /2 525
|/2 ..... 99, 231, 383

~

~ /1 525
~/.bashrc 22, 1102, 1104
~/.ciaorc 211
~/.cshrc 21, 1101, 1104
~/.emacs 22, 1102, 1104, 1112

+

+ /1 156
+ /2 157
+/1 ..... 350, 371, 373, 546, 547, 571
+/2 ..... 350, 371, 373, 374, 546, 547, 571, 572
++ /1 157
++ /2 527

>

>/2 ..... 151, 154
>=/2 ..... 151, 155
>> /2 157

^

^ /1 526
^/2 ..... 212, 233, 234, 236, 472
^^ /1 527

\

\ /1 157
\/ /2 157
\=/2 129
\==/2 ..... 135, 136
\+/1 ..... 99, 100

<

<#/2 605
<&!/1 543
<&&!/1 543
<&&/1 543
<&/1 543
<-/1 ..... 481, 594
<-/2 ..... 481, 594, 673
</2 ..... 151, 153
<=/2 481
<?/1 543
<< /2 157
<v>libroot</v>/ciao/DOTcshrc 1106

A

A. Ciepielewski 49, 59
a_string/1 654
abolish/1 ..... 213, 239, 242
abort 56
abort/0 ..... 177, 179
abs/1 157
absolute_file_name/2 ..... 159, 164, 301
absolute_file_name/7 ..... 159, 165
abstract methods 620
acceptable modes 350
accepted_type/2 739
access_benchmark_data/8 388
ACCLAIM 10
acknowledgments 9
acrobat reader 24, 29
active module 37, 46, 585
active modules 9, 507
active object 585
activemod 69
actmods 585
acyclic_term/1 449
add_after/4 ..... 283, 290, 787
add_assoc/4 ..... 777, 783

add_before/4 ... 283, 290, 787, 788
add_clause_trans/2 ... 195
add_edges/3 ... 797, 798
add_environment_whitespace/3 ... 1049, 1055
add_goal_trans/1 ... 195
add_goal_trans/2 ... 195
add_indentation/3 ... 1049, 1056
add_lines/4 ... 197, 200
add_name_value/2 ... 487, 489
add_preffix/3 ... 493, 498
add_sentence_trans/1 ... 193, 195
add_sentence_trans/2 ... 194
add_suffix/3 ... 493, 498
add_term_trans/1 ... 195
add_term_trans/2 ... 194
add_vertices/3 ... 797, 798
add_vpath/1 ... 487, 490
add_vpath_mode/3 ... 487, 490
addmodule and pred(N) meta-arguments ... 617
address/1 ... 661, 662
agent ... 589
aggr/1 ... 605, 606
aggregates ... 211, 212, 213, 231, 233, 383, 388, 471, 487, 669, 675, 685, 725, 1005, 1007, 1009, 1021, 1039, 1041, 1047, 1049, 1089
aggregation predicates ... 236
alias_file/1 ... 427, 428
all_values/2 ... 487, 488
Amadeo Casas ... 525, 543, 545, 993
AMOS ... 10
analyzer output ... 347
ancestors ... 55
andorra ... 539
andprolog ... 543
andprolog/andprolog_rt ... 543
Angel Fernandez Pineda ... 423, 457, 461, 611, 617, 629, 635, 641, 997
Anil Nair ... 511
Anne Mulkers ... 10
answer variable ... 42
any_term/1 ... 661, 662
any_to_term/2 ... 493, 499
apll ... 545
apll/visandor ... 546
append/2 ... 12, 395
append/3 ... 211, 283
apply_vpath_mode/4 ... 487, 491
apropos/1 ... 457, 459
apropos_spec/1 ... 460
aref/3 ... 775
arefa/3 ... 775
arefl/3 ... 775, 776
arg/2 ... 441, 442
arg/3 ... 129, 130
arg_expander/6 ... 461, 462
argnames ... 521
argnames/1 ... 521, 522, 673
argspec/1 ... 513
arithexpression/1 ... 151, 156
arithm_average/2 ... 388, 390
arithmetic ... 151, 341
array_to_list/2 ... 775, 776
arrays ... 775
Arsen Kostenko ... 927, 929, 935, 937, 939, 941, 953, 957, 961, 971, 975, 979
ASAP ... 10
asbody_to_conj/2 ... 435, 438
ASCII code ... 157
aset/4 ... 775, 776
ask/2 ... 445
assert/1 ... 214, 239, 240
assert/2 ... 214, 239, 241
assert_body_type/1 ... 435, 438
asserta/1 ... 214, 239
asserta/2 ... 214, 239, 240, 244
asserta_fact/1 ... 187, 188, 723, 725, 726, 735
asserta_fact/2 ... 187
assertion body syntax ... 349, 352, 353
assertion language ... 3
assertion language ... 6
assertion language ... 74
assertion normalizer ... 821
assertion status ... 341, 342, 345
assertions ... 67, 75, 339, 340, 349
assertions/assertions_props ... 340, 357, 455
assertions/assrt_lib ... 455, 669
assertions/c_itf_props ... 825
assertions/doc_props ... 275, 277
assertions/native_props ... 99, 105, 121, 129, 135, 141, 151, 169, 197, 233, 249, 283, 423, 441, 471, 511, 517
assertions_props ... 349
assertz/1 ... 214, 239, 240
assertz/2 ... 214, 239, 240, 244
assertz_fact/1 ... 187, 188, 723, 725, 726, 727, 735
assertz_fact/2 ... 187, 188

assoc ... 777
assoc_to_list/2 ... 777
assrt_body/1 ... 340, 349
assrt_status/1 ... 349, 353
assrt_type/1 ... 349, 354
assrt_write ... 455
at_least_one/4 ... 1049
at_least_one/5 ... 1049
atan/1 ... 157
atm/1 ... 105, 108, 929, 930, 931, 932, 933
atm_or_atm_list/1 ... 105, 115
atm_or_int/1 ... 159, 167
atom/1 ... 121, 122
atom_chars/2 ... 217, 259
atom_codes/2 ... 141, 142
atom_concat/2 ... 441, 442
atom_concat/3 ... 141, 147
atom_length/2 ... 141, 147
atom_lock_state/2 ... 465, 469
atom_number/2 ... 141, 143
atom_number/3 ... 141, 146
atom_or_str/1 ... 423, 425
atom2term ... 413
atom2term/2 ... 413, 414
atomic/1 ... 121, 125
atomic_basic ... 141
AtomName/Arity ... 393
attach_attribute/2 ... 201
attribute ... 618
attributed variables ... 201
attributes ... 133, 201
attributes/1 ... 883, 884, 890, 893
Austrian Research Institute for AI ... 10
auto-documenter command args, setting ... 78
auto-documenter command, setting ... 78
auto-documenter default format, setting ... 77
auto-documenter lib path, setting ... 78
auto-documenter working dir, setting ... 78
auto-fill ... 67
auto-indentation ... 67
axis_limit/1 ... 865, 866, 868, 872

B

backup file ... 724
bagof/3 ... 213, 233, 234, 471, 472
baktrak1/1 ... 385
baktrak2/1 ... 385
barchart1/10 ... 876
barchart1/7 ... 852, 865
barchart1/8 ... 875
barchart1/9 ... 852, 865, 866
barchart2/10 ... 880
barchart2/11 ... 852, 871, 872
barchart2/7 ... 852, 871
barchart2/8 ... 879
barchart3/7 ... 852, 875
barchart3/9 ... 852, 875, 876
barchart4/11 ... 853, 879, 880
barchart4/7 ... 853, 879
basename/2 ... 423, 425
bash ... 22, 35, 87, 1102, 1104, 1107
basic_props ... 105, 547
basic_props.pl ... 377
basic_props:regtype/1 ... 355
basiccontrol ... 99
basicmodes ... 373
basictypes ... 739
benchmark/6 ... 1083, 1084
benchmark_usage/1 ... 388
benchmark2/6 ... 1083, 1085
benchmarks/benchmark_utilities ... 388
benchmarks/boresea ... 388
benchmarks/choice ... 388
benchmarks/cut ... 388
benchmarks/deref ... 388
benchmarks/envir ... 388
benchmarks/index ... 388
benchmarks/results ... 388
benchmarks/small_programs ... 388
benchmarks/unif ... 388
between ... 211, 219, 265, 299
between/3 ... 219, 299
BeyondInstall ... 1115
bf ... 593, 596
binary directory ... 1103
bind_ins/1 ... 105, 119
bind_socket/3 ... 473, 474
bind_socket_interface/1 ... 763
block ... 845
block/1 ... 845
bltclass ... 857
bltwish_interp/1 ... 857
body/1 ... 452
body_expander/6 ... 461, 462
body2list/2 ... 435, 438

bold_message/1 ..... 487, 490
bold_message/2 ..... 487, 490
bolder_message/1 ..... 487, 490
bolder_message/2 ..... 487, 490
bound/1 1035
bound_double/1 1035
boundary 1005
boundary_check/3 1005
boundary_rotation_first/2 1005
boundary_rotation_last/2 ..... 1005, 1006
bounds/3 ..... 982, 985
box-type debugger 49
breadth first execution 596
breadth-first execution 9, 507
breakpoins 51
breakpoints 49, 72
breakpt/6 ..... 50, 51, 59
Bristol University 10
browse/2 ..... 457, 458
buffer 68
bugs, reporting 1116
build_foreign_interface 669
build_foreign_interface/1 ..... 669, 670
build_foreign_interface_explicit_decls/1 .. 670
build_foreign_interface_explicit_decls/2 .. 669, 670
build_foreign_interface_object/1 ..... 669, 670
building standalone distributions 827
builtin directives 103, 193
builtin modules 91
builtin_directives 103
byrd-box model 72
byte/1 661
byte_list/1 ..... 646, 661, 662
bytecode object files 1103

C

C 1103
C/3 ..... 129, 133
c:/.emacs 1112
c_assrt_body/1 ..... 349, 352
c_errno/1 ..... 223, 301, 305, 505
c_itf 461
c_itf_internal 410
cache 733
call/1 ..... 265, 352, 515, 735, 736
call/2 ..... 95, 515
call/N 247, 515
call_graph/2 825
call_in_module/2 ..... 56, 59
call_unknown/1 ..... 487, 488
callable/1 ..... 105, 110
callgraph 825
Calling emacs 9, 643
calls assertion 341
calls/1 ..... 340, 341, 344
calls/2 ..... 340, 341
cancellation/1 ..... 546, 562
canonic_html_term/1 ..... 715, 716, 717
canonic_xml_item/1 1096
canonic_xml_query/1 1096
canonic_xml_subquery/1 1097
canonic_xml_term/1 ..... 715, 716, 717, 1096
case_insensitive_match/2 791
cat/2 ..... 493, 494
cat_append/2 ..... 493, 494
catch/3 ..... 177, 178
cd/1 ..... 23, 28, 222, 301, 310, 503
ceiling/1 157
cell_value/1 ..... 906, 907
certainty factor 997
CGI 701, 703
CGI executables 63
change, author 75
change, comment 75
changelog 67
changelog entry 75
changing the executables used 78
char_code/2 ..... 217, 259
char_codes/2 ..... 217, 259, 260
char_conversion/2 193
character string 339
character_code/1 ..... 105, 114
character_count/2 ..... 159, 162
chartlib 847
chartlib/bltclass 859, 865, 871, 875, 879, 884, 893, 901, 905, 909, 911, 913
chartlib/chartlib_errhandle 852
chartlib/color_pattern ... 865, 871, 875, 879, 884, 893, 901
chartlib/genbar1 ..... 852, 871, 875, 879, 884, 893, 901, 905, 909, 911, 913
chartlib/genbar2 852
chartlib/genbar3 852
chartlib/genbar4 852

chartlib/gengraph1 ..... 852, 893
chartlib/gengraph2 852
chartlib/genmultibar 852
chartlib/install_utils ... 859, 865, 871, 875, 879, 884, 893, 901, 905, 909, 911, 913
chartlib/table_widget1 ..... 852, 909, 911, 913
chartlib/table_widget2 852
chartlib/table_widget3 852
chartlib/table_widget4 852
chartlib/test_format 865, 871, 875, 879, 884, 893, 901, 905, 909, 911, 913
chartlib_errhandle 855, 859
chartlib_text_error_protect/1 ..... 855, 859, 915
chartlib_visual_error_protect/1 ..... 855, 859
check assertion 346
Check(X) 364
check/1 ..... 340, 346, 347
check_sublist/4 ..... 915, 916
check_var_exists/1 ..... 487, 489
checking the assertions 73
children_nodes/1 ..... 1005, 1006
chmod/2 ..... 220, 301, 317, 501
chmod/3 ..... 220, 301, 317, 501
choice_point/1 385
choose_free_var/2 ..... 982, 984
choose_value/2 ..... 982, 984
choose_var/3 ..... 982, 983
choose_var_nd/2 ..... 982, 984
Christian Holzbauer 10
Christian Holzbaur 201, 597, 601
ciao 21, 24, 29, 80, 87, 1103, 1105, 1111
Ciao basic builtins 8, 89
Ciao engine 10, 34, 36
Ciao engine builtins 167
Ciao mode version 79
Ciao preprocessor 10, 67, 73
Ciao Prolog 391
Ciao scripts 1103
Ciao shell 1103
Ciao top-level 67
ciao, global description 3
Ciao, why this name 5
ciao-mode-init 1104
ciao-shell 21, 63, 64, 65, 1103, 1105
ciao-users 1115
ciao.el 1104
ciao.reg 1113, 1114
ciao_c_headers_dir/1 ..... 205, 207
ciao_flag/3 ..... 182, 184
ciao_lib_dir/1 ..... 205, 208
ciaoc 21, 23, 28, 33, 35, 38, 65, 103, 1102, 1103, 1105, 1108
ciaoc.bat 33
CiaoMode 67
ciaopaths 383
ciaopp 1, 7, 10, 14, 31, 36, 49, 67, 73, 80, 359
CiaoPP 647
ciaosh 7, 21, 27, 31, 41, 65, 67, 457, 1105, 1108, 1112
ciaosh.cpx 1111
cl_option/2 ..... 391, 392
class 617
class constructor 635
class instances 635
class_doc 641
class_name/1 ..... 635, 638
class_source/1 ..... 635, 639
Claudio Ochoa 987
Claudio Vaucheret 539, 595, 605
clause/1 452
clause/2 ..... 213, 239, 242
clause/3 213
clauses/1 452
clean_imperative_guard/2 680
clean_measures/0 ..... 546, 568
clearerr/1 ..... 159, 163
client installation 1113
client.bat 1113
CLIP group 9
clique/1 359
clique_1/1 359
clockfreq_option/1 ..... 225, 323, 326
clockfreq_result/1 ..... 226, 323, 325
close/1 ..... 159, 160, 311
close/2 267
close_client/0 771
close_DEF/5 ..... 1023, 1032
close_EXTERNPROTO/6 ..... 1023, 1031
close_file/1 ..... 226, 329, 330
close_input/1 397
close_node/5 ..... 1023, 1029
close_nodeGut/4 ..... 1023, 1030
close_output/1 ..... 397, 398
close_predicate/1 ..... 187, 190
close_PROTO/6 ..... 1023, 1030
close_Script/5 ..... 1023, 1033

closed 188, 189, 190
clp_entailed/1 599
clp_meta/1 599
clpq 533, 597
clpr 601
clterm/1 452
CM 10
code_class/2 ..... 169, 170, 173
collect_singletons/2 395
collecting files which are modules 841
color/1 ..... 861, 890
color/2 ..... 861, 862
color_pattern 861
coloring, syntax 67
combine_attributes/2 ..... 201, 202, 203
command 56
comment assertion 346
comment string 350, 352, 353
comment/2 ..... 75, 340, 346
comments, machine readable 339
comp assertion 343
comp/1 ..... 340, 343, 353
comp/2 ..... 340, 343
comparator/1 ..... 135, 138
compare/3 ..... 135, 138, 472
compare_benchmark/7 ..... 1083, 1084
compare_benchmark2/7 ..... 1083, 1085
compat/2 ..... 105, 116
compatibility properties 355
compatible 349
compile the whole Ciao distribution 839
compile/1 ..... 23, 28, 43, 45, 79
compiler 212, 297, 1102
compiler, standalone 33
compiler/c_itf 43, 461, 669
compiler/c_itf_internal 297, 405
compiler/compiler 43, 211, 487, 635, 759
compiler/engine_path 669
compiler/exemaker 43, 297
compiler_and_opts/2 667
compiling 69, 70
compiling programs 22, 23, 28, 29
compiling, from command line 33
compiling, Win32 1107
complete proof procedure 595
complete_dict/3 ..... 813, 814
complete_vars_dict/3 ..... 813, 814
complex argument property 349, 350, 352, 353
complex goal property 349, 351, 353
complex_arg_property/1 ..... 349, 350, 352, 353
complex_goal_property/1 ..... 349, 351, 353
compound/1 ..... 218, 265
computational cost 73
conc_aggregates 471
concurrency 465
concurrency/concurrency .. 752, 759, 763, 937, 939, 971
concurrent 189, 190, 469
concurrent attribute 619
concurrent predicate 187, 190, 191, 192
concurrent predicates 187
concurrent updates 723
concurrent/1 ..... 187, 192, 465, 469, 470, 617, 619
condcomp 209
configuration file 831
conj_disj_type/1 ..... 435, 438
conj_to_list/2 ..... 435, 436
conj_to_llist/2 ..... 435, 437
connect_to_socket/3 ..... 473, 474
connect_to_socket_type/4 ..... 473, 474, 477
consistent_hash/2 ..... 957, 958
const_head/1 ..... 129, 133
constant/1 ..... 105, 110
constraint logic programming 9, 507
constraint/1 ..... 359, 360
constructor 621
constructor/0 ..... 617, 621
constructor/1 ..... 635, 638
consult/1 ..... 23, 28, 43, 45, 79
contains_ro/2 ..... 283, 291
contains1/2 ..... 283, 291
Context-sensitive 22, 28
continue/3 1063
contributed libraries 9, 843
control 577
convert_atoms_to_string/2 1039
convert_permissions/2 ..... 493, 499
convert_permissions/4 ..... 493, 499
copy_args/3 441
copy_file/2 ..... 223, 301, 305, 505
copy_file/3 ..... 223, 301, 305, 505
copy_files/2 ..... 493, 494
copy_files/3 ..... 493, 494
copy_files_nofail/3 ..... 493, 494
copy_stdout/1 429
copy_term/2 ..... 129, 132, 133, 449

copy_term_nat/2 ..... 129, 133
core/1 700
correct_commenting/4 ..... 1049, 1052
cos/1 157
cost/3 ..... 1083, 1086
counters 735, 785
covered/1 ..... 359, 360
covered/2 ..... 359, 360
cp_name_value/2 ..... 487, 489
create/2 ..... 725, 728
create_dict/2 813
create_dictionaries/1 1009
create_directed_field/5 ..... 1049, 1052
create_environment/4 ..... 1049, 1054
create_field/3 ..... 1049, 1051
create_field/4 ..... 1049, 1051
create_field/5 ..... 1049, 1051
create_from_list/2 919
create_node/3 ..... 1049, 1051
create_parse_structure/1 ..... 1049, 1053
create_parse_structure/2 ..... 1049, 1053
create_parse_structure/3 ..... 1049, 1053
create_pretty_dict/2 813
create_proto_element/3 1041
creating executables 69
creep 54
cross_product/2 ..... 12, 283, 294
csh 21, 87, 1101, 1104
csh-compatible shell 21, 1101, 1104
ctrlc_clean/1 417
ctrlcclean 417, 669
ctrlcclean/0 417
CUBICO 10
current input 246
current input stream 161
current output stream 161, 162
current_atom/1 ..... 224, 323, 327
current_ciao_flag/2 ..... 182, 184
current_env/2 ..... 224, 301, 304, 505
current_executable/1 ..... 222, 301, 308, 504
current_fact/1 ..... 187, 188, 190, 735, 736
current_fact/2 ..... 187, 189
current_fact_nb/1 ..... 187, 190
current_host/1 ..... 222, 301, 308, 504
current_infixop/4 ..... 255, 256
current_input/1 ..... 159, 161, 162
current_key/2 ..... 226, 331, 332
current_module/1 ..... 205, 207
current_op/3 ..... 211, 255, 256, 257
current_output/1 ..... 159, 162
current_postfixop/3 ..... 255, 257
current_predicate/1 ..... 213, 239, 242
current_predicate/2 ..... 213, 239, 243
current_prefixop/3 ..... 255, 256
current_prolog_flag/2 182
current_stream/3 ..... 159, 163
customize 68, 78
cuttest/1 387
cyclic_term/1 449
cyclic_terms 449
cyg2win/3 ..... 219, 301, 319, 501
Cygnus Win32 1107
Cygwin 11

D

D.H.D. Warren 10
D.L. Bowen 10
Daniel Cabeza ... 9, 33, 41, 49, 59, 63, 65, 67, 91, 97, 99, 103, 105, 121, 129, 135, 151, 159, 169, 181, 187, 193, 197, 201, 205, 211, 233, 239, 245, 249, 255, 259, 265, 275, 301, 323, 335, 371, 401, 413, 421, 423, 433, 449, 473, 515, 517, 521, 525, 577, 579, 585, 593, 597, 601, 701, 703, 713, 715, 723, 807, 823, 827, 841, 1101, 1111, 1115
data 243
data declaration 187
data file 724
data predicate 187, 188, 189, 190, 191, 192
data/1 ... 187, 192, 213, 239, 243, 521, 617, 618, 619, 624, 724
data_facts 187
data_facts:asserta_fact/1 735
data_facts:assertz_fact/1 735
data_facts:current_fact/1 736
data_facts:retract_fact/1 736
database initialization 727
datime/1 ..... 224, 301, 302, 506
datime/9 ..... 224, 301, 302, 506
datime_atom/1 ..... 493, 498
datime_atom/2 ..... 493, 498
datime_string/1 ..... 493, 498
datime_string/2 ..... 493, 498
datime_struct/1 ..... 224, 301, 304, 506
David H.D. Warren 233, 471
David Trallero Mena 919, 989, 1077, 1083

davinci 685
davinci/0 685
davinci_command/1 686
davinci_get/1 685
davinci_get_all/1 685
davinci_lgraph/1 ..... 685, 686
davinci_put/1 ..... 685, 686
davinci_quit/0 ..... 685, 686
davinci_ugraph/1 ..... 685, 686
dbId/2 743
dcg 271
dcg_tr 275
dcg_translation/2 275
ddlist 919
ddlist/1 ..... 919, 923
ddlist_member/2 ..... 919, 923
debug 50
debug (interpreted) mode 49
debug options 54
debug/0 ..... 50, 59
debug/1 ..... 197, 198, 199
debug_goal/2 ..... 405, 408
debug_goal/3 ..... 405, 409
debug_message/1 ..... 405, 408
debug_message/2 ..... 405, 408
debug_module/1 ..... 50, 59
debug_module_source/1 ..... 50, 60
debugger 49, 50, 59
debugger/debugger 43, 99
debugger/debugger_lib 59
debugger_lib 59, 60, 61
debugging 55, 72
debugging tools 49
debugging, source-level 67, 72
debugging/0 60
debugrtc/0 60
dec_indentation/2 ..... 1049, 1056
dec10_io 211, 226, 329, 388
decl assertion 345
decl/1 ..... 340, 345, 349
decl/2 ..... 340, 345
declarations, user defined 103
decompose_field/3 ..... 1023, 1033
DECsystem-10 Prolog User's Manual 10
deductive database 723
default constructor 632, 636
default_for_ciaosh 41
default_predicates 211, 745
define_flag/3 .. 59, 61, 182, 186, 245, 247, 249, 254, 301, 320, 703, 711, 1073, 1076
del_assoc/4 ..... 777, 783
del_dir_if_empty/1 493
del_edges/3 ..... 797, 798
del_endings_nofail/2 ..... 493, 495
del_env/1 ..... 223, 301, 305, 505
del_file_nofail/1 ..... 493, 495
del_file_nofail/2 ..... 493, 495
del_files_nofail/1 ..... 493, 495
del_global/1 537
del_max_assoc/4 ..... 777, 784
del_min_assoc/4 ..... 777, 784
del_name_value/1 ..... 487, 489
del_vertices/3 ..... 797, 798
delaying predicate execution 8, 507
delete/1 ..... 697, 700
delete/2 ..... 919, 921
delete/3 ..... 211, 283, 287, 787, 788
delete_after/2 ..... 919, 921
delete_directory/1 ..... 219, 301, 318, 501
delete_file/1 ..... 219, 301, 317, 501
delete_files/1 ..... 493, 495
delete_non_ground/3 ..... 12, 283, 287
delete_on_ctrlc/2 417
delete_top/2 ..... 919, 921
dependent files 827
deprecated/1 ..... 105, 117
depth first iterative deepening 595
depth limit 595
derived_from/2 ..... 635, 637
describe/1 ..... 457, 459
destroy/1 ..... 635, 638
destructor 622
destructor/0 ..... 617, 622
det_hook 571
det_hook/det_hook_rt 571
det_hook_rt 575
det_try/3 575
detach_attribute/1 ..... 201, 202
detcond/1 ..... 539, 540
determinacy 73
determinate goal 539
determinate/2 539
development environment .. 24, 29, 1102, 1105, 1111
dgraph/1 793
dgraph_to_ugraph/2 793
dht/dht_config 935, 937, 939, 941, 957, 961

dht/dht_logic 937, 939
dht/dht_logic_misc 935, 937, 941
dht/dht_misc 929, 937, 939, 961
dht/dht_routing 941
dht/dht_rpr 941
dht/dht_s2c 935
dht/dht_s2s 935
dht/dht_storage 941
dht_check_predecessor 942
dht_check_predecessor/1 ..... 941, 942
dht_client 929
dht_closest_preceding_finger/2 .... 941, 942, 943
dht_config 975
dht_config.pl 935, 937, 939, 957
dht_connect/2 ..... 929, 930, 931
dht_connect/3 929
dht_connection/2 ..... 929, 930, 931, 932, 933
dht_consult/4 ..... 930, 931
dht_consult_b/2 ..... 971, 972
dht_consult_b/4 ..... 929, 930
dht_consult_nb/2 ..... 971, 973
dht_consult_nb/4 ..... 929, 931
dht_consult_server_b/3 ..... 941, 946, 949, 950
dht_consult_server_nb/3 ..... 941, 947, 948
dht_disconnect/1 ..... 929, 930
dht_extract_b/2 ..... 971, 972
dht_extract_b/4 ..... 929, 931
dht_extract_from_server_b/3 ..... 941, 949
dht_extract_from_server_nb/3 ..... 941, 950
dht_extract_nb/2 ..... 971, 972, 973
dht_extract_nb/4 ..... 929, 931
dht_find_and_*/2 ..... 946, 947, 951
dht_find_and_consult/2 ..... 947, 948, 949, 950
dht_find_and_consult_b/2 ..... 941, 946
dht_find_and_consult_nb/2 ..... 941, 947
dht_find_and_extract/2 ..... 949, 950
dht_find_and_extract_b/2 ..... 941, 948
dht_find_and_extract_nb/2 ..... 941, 950
dht_find_and_store 951
dht_find_and_store/2 ..... 941, 951
dht_find_predecessor/2 ..... 941, 943, 944
dht_find_successor/2 ..... 941, 944
dht_finger/1 942
dht_finger/2 ..... 941, 942
dht_finger_start/2 ..... 953, 954
dht_finger_table/2 953
dht_fix_fingers/0 ..... 941, 945, 946
dht_hash/3 ..... 929, 932
dht_id_by_node/2 ..... 941, 946
dht_init/1 941
dht_join/1 ..... 941, 944, 945
dht_join/2 944
dht_join_host/1 ..... 975, 976
dht_key_hash 974
dht_key_hash/2 ..... 971, 973
dht_logic 941
dht_logic.pl 953, 957
dht_logic_misc 957
dht_misc 979
dht_notify/1 ..... 941, 944, 945
dht_predecessor/1 ..... 953, 956
dht_prolog/1 935
dht_reset_predecessor/0 ..... 953, 956
dht_routing 953
dht_routing.pl 946
dht_rpr 961
dht_rpr.pl 946
dht_rpr_call/2 ..... 961, 965, 968
dht_rpr_call/3 ..... 961, 968, 969
dht_rpr_clear_by_node/1 ..... 961, 964
dht_rpr_clear_node 964
dht_rpr_compose_id/3 ..... 961, 964
dht_rpr_id_by_node/2 ..... 946, 961, 962
dht_rpr_node/1 ..... 961, 965
dht_rpr_node_by_id/2 ..... 961, 962, 963
dht_rpr_node_id/1 ..... 961, 963
dht_rpr_register_node/1 961
dht_rpr_register_node/2 961
dht_s2c 937
dht_s2c.pl 937
dht_s2c:dht_s2c_mian/0 935
dht_s2c_main/0 937
dht_s2c_port/1 975
dht_s2c_threads/1 ..... 975, 976
dht_s2s 939
dht_s2s:dht_s2s_mian/0 935
dht_s2s_main/0 939
dht_s2s_port/1 ..... 975, 976
dht_s2s_threads/1 ..... 975, 976
dht_server 935
dht_server.pl 937, 939
dht_server/1 935
dht_server_host/1 ..... 975, 977
dht_server_id/1 ..... 975, 977
dht_set_finger/4 ..... 953, 955
dht_set_hash_power/1 975

dht_set_join_host/1 ..... 975, 977
dht_set_predecessor/1 ..... 953, 956
dht_set_s2c_port/1 ..... 975, 976
dht_set_s2c_threads/1 ..... 975, 976
dht_set_s2s_port/1 ..... 975, 976
dht_set_s2s_threads/1 ..... 975, 976
dht_set_server_host/1 ..... 975, 977
dht_set_server_id/1 ..... 975, 977
dht_stabilize/0 ..... 941, 942, 944, 945
dht_storage 971
dht_store/3 971
dht_store/4 ..... 929, 932
dht_store_to_server/4 ..... 941, 951
dht_successor/1 ..... 941, 942
dht_update_finger/2 ..... 953, 955
dic_get/3 ..... 399, 400
dic_lookup/3 399
dic_lookup/4 ..... 399, 400
dic_node/2 399
dic_replace/4 ..... 399, 400
dict 399
dict2varnamesl/2 ..... 813, 814
dictionary 1007
dictionary/1 ..... 349, 352, 399, 1035
dictionary/5 399
dictionary/6 1007
dictionary_insert/5 ..... 1009, 1010
dictionary_lookup/5 ..... 1009, 1010
dictionary_tree 1009
diff_vars/3 447
difference/3 ..... 283, 293
dir_path/2 ..... 223, 301, 305, 505
directives 103
directory_files/2 ..... 221, 301, 313, 502
directoryname/1 ..... 725, 728
DISCIPL 10
discontiguous/1 ..... 103, 617
disj_to_list/2 ..... 435, 437
disj_to_llist/2 ..... 435, 438
display 55
display/1 ..... 169, 175, 197, 198, 199, 249
display/2 ..... 169, 174, 175, 249
display_list/1 ..... 197, 199
display_string/1 ..... 197, 198, 199
display_term/1 ..... 197, 199
displayq/1 ..... 169, 175, 197, 198
displayq/2 ..... 169, 175
distributed execution 585
Distributed Programming Model 747
div_times/2 ..... 1083, 1086
dlgraph/1 793
dlgraph_to_lgraph/2 ..... 793, 794
dlist/3 ..... 283, 290
do/2 ..... 493, 496
do/3 ..... 493, 496
do/4 ..... 493, 496
do/5 ..... 493, 496, 497
do_atmlist__popen/2 ..... 493, 500
do_interface/1 ..... 669, 671
do_not_free/2 ..... 646, 661, 663
do_on_abolish/1 ..... 239, 244
do_options/1 ..... 493, 497, 500
do_str/3 ..... 493, 500
do_str_without_nl/3 ..... 493, 500
do_str_without_nl__popen/2 ..... 493, 500
doc/2 ..... 340, 345, 346
docstring/1 ..... 339, 349, 350, 352, 353, 354
documentation generator 7, 31
dot_concat/2 ..... 487, 488
double_list/1 661
DOVES 10
downloading emacs 1105, 1111
downloading new versions 9, 1099
downloading, latest versions 1115
dump_benchmark_data/0 388
dump_constraints/3 599
dvips 831
dyn_load_cfg_module_into_make/1 ..... 487, 491
dynamic 211, 213, 214, 231, 243, 383, 1005, 1007, 1009, 1021, 1039, 1041, 1047, 1049
dynamic predicate 187
dynamic/1 ..... 213, 239, 243, 617, 619
dynamic_clauses 239
dynamic_rt 239
dynamic_search_path/1 ..... 43, 46

E

ecrc 385
edges/2 797
edges_to_lgraph/2 ..... 793, 794
edges_to_ugraph/2 ..... 793, 794
EDIPIA 10
Edison Mera 33, 49, 59, 197, 259, 359, 377, 379, 487, 517, 813, 999
elisp_string/1 ..... 768, 769

ELLA 10
emacs .. 3, 21, 22, 24, 27, 28, 29, 30, 49, 52, 53, 54, 64, 67, 68, 70, 73, 75, 76, 79, 80, 85, 767, 768, 769, 1101, 1102, 1104, 1105, 1106, 1111, 1112, 1113
emacs Ciao mode 75
emacs interface 7, 31, 41, 67
emacs lisp 767
emacs menu bar 68
emacs mode 49, 64, 67, 767
emacs mode setup 1104
emacs mode, loading several 79
emacs mode, setting up, Win32 1112
emacs server 767
emacs, download 1105, 1111
emacs, intro 24, 29
emacs_edit/1 768
emacs_edit_nowait/1 768
emacs_eval/1 768
emacs_eval_nowait/1 768
embedded debugger 49, 51, 79
empty_assoc/1 777
eng_backtrack/2 ..... 465, 466
eng_call/3 ..... 465, 466
eng_call/4 ..... 465, 466
eng_cut/1 ..... 465, 466
eng_goal_id/1 ..... 465, 468
eng_kill/1 ..... 465, 467
eng_killothers 467
eng_killothers/0 ..... 465, 467
eng_release/1 ..... 465, 466, 467
eng_self/1 ..... 465, 467
eng_status/0 ..... 465, 468
eng_wait/1 ..... 465, 467
engine 1103
engine directory 8, 89
engine module 509
Enrico Pontelli 10
ensure_loaded/1 .... 34, 35, 43, 44, 45, 97, 212, 297
ensure_loaded/2 297
enter_mutex/1 ..... 546, 567
enter_mutex_remote/1 ..... 546, 568
enter_mutex_self/0 ..... 546, 568
entry assertion 344
entry/1 ..... 340, 344, 352
envir0ar/1 386
environment variable 1103
environment variable definitions 1101, 1104
environment variables 87, 1104
environment variables, setup 21
environment/1 ..... 1035, 1036
eq/3 ..... 675, 681
equal_lists/2 ..... 283, 293
equalnumber/3 915
equiv/2 ..... 105, 118
erase/1 ..... 187, 191, 244
errhandle 419, 669
error term 181
error(a,b) 379
error(unintercepted_signal(Signal), send_signal/1-1) 178
error/1 ..... 197, 198
error_file/2 860
error_free/1 ..... 105, 119
error_message/1 405
error_message/2 ..... 405, 860
error_message/3 405
error_protect/1 419
error_vrml/1 1013
etags/2 ..... 493, 499
etc 1103, 1113
eval/1 ..... 105, 118
evaluable functors 156
examples 23, 28, 29
examples/webbased_server/webbased_server.pl 587, 588
exception(error(a,b)) 379
exception/1 ..... 359, 360
exception/2 ..... 359, 360
exceptions 177, 649
exec('ls -lRa ../sibling_dir', In, Out, Err) 312
exec/3 ..... 221, 301, 312, 503
exec/4 ..... 221, 301, 312, 321, 503
exec/8 ..... 221, 301, 312, 313, 502
executable 33
executables 1103
executables, compressed 37
executables, dynamic 35
executables, generating 23, 28
executables, how to run 34
executables, lazy load 36
executables, self-contained 36
executables, static 36
executables, types 35
execute_permissions/2 ..... 493, 499
execute_permissions/4 ..... 493, 499

execution visualizers 49
exit assertion 344, 345
exit/1 ..... 340, 344, 345
exit/2 ..... 340, 344
exit_mutex/1 ..... 546, 568
exit_mutex_remote/1 ..... 546, 568
exit_mutex_self/0 ..... 546, 568
exp/1 157
expander_pred/1 462
expansion 193
expansion_tools 461
expansions 7
Explorer 27, 1112
export/1 ..... 92, 617, 618, 997
exports 377, 821
extensibility 4
extension/2 ..... 423, 425
External interface 9, 643
extra_compiler_options 664
extra_compiler_opts/1 664
extra_compiler_opts/2 664
extra_linker_options/1 665
extra_linker_opts/1 664
extra_linker_opts/2 665
extract_paths/2 ..... 223, 301, 306, 505

F

F.C.N. Pereira 10
facts 735, 736
facts/2 737
factsdb 733, 735
factsdb/factsdb_rt 733
factsdb_rt 733, 735
faggregator/1 ..... 605, 608
fail 55
fail/0 ..... 99, 101
fails/1 ..... 359, 361
false assertion 347
false/0 ..... 99, 102
false/1 ..... 340, 347
fast_read/1 ..... 421, 422
fast_read/2 ..... 421, 422, 979
fast_write/1 ..... 421, 422
fast_write/2 ..... 421, 422, 979
fast_write_to_string/3 ..... 421, 422
fastrw 421, 457, 763, 771, 979
fd 981
fd_item/1 982
fd_range/1 982
fd_store/1 ..... 982, 983
fd_store_entity/1 ..... 982, 983
fd_subrange/1 982
feature terms 8, 507, 521
fetch_url/3 713
field_Id/1 ..... 1047, 1048
field_type 1015
field_value 1017
field_value_check 1019
fieldType/1 1015
fieldValue/6 1017
fieldValue_check/8 1019
file_alias 737
file_alias/2 ..... 427, 428, 735, 737
file_dir_name/3 ..... 223, 301, 306, 505
file_directory_base_name/3 ..... 423, 424
file_exists/1 ..... 220, 301, 314, 502
file_exists/2 ..... 220, 301, 314, 502
file_locks 433
file_locks/file_locks 725
file_name_extension/3 ..... 423, 424
file_properties/6 ..... 220, 301, 315, 502
file_property/2 ..... 220, 301, 315, 502
file_search_path/2 ..... 34, 35, 65, 159, 166, 167
file_terms/2 429
file_to_string/2 429
file_to_string/3 ..... 429, 430
file_utils 359, 429, 493
filed predicate 733
fileerrors/0 ..... 182, 185
fileinfo 821
filenames 405, 423, 487, 669
fillout/4 ..... 1049, 1050
fillout/5 ..... 1049, 1050
filter/2 ..... 105, 119
filter_alist_pattern/3 ..... 493, 495
find_det_goal/2 ..... 546, 561
find_file/2 ..... 487, 489
find_goal/2 ..... 546, 561
find_name/4 ..... 813, 815
find_server/3 ..... 946, 947
findall/3 ..... 212, 233, 234, 235, 236, 471
findall/4 ..... 212, 233, 235
findnsols/4 ..... 212, 233, 235, 236
findnsols/5 ..... 212, 233, 236
finite_solutions/1 ..... 359, 361

first-timers ... 211
flag/1 ... 452
flag_values/1 ... 105, 119
flatten/2 ... 395
float/1 ... 121, 124, 157
float_fractional_part/1 ... 157
float_integer_part/1 ... 157
floor/1 ... 157
flt/1 ... 105, 107, 156
flush_output/0 ... 159, 163
flush_output/1 ... 159, 163
fmode/2 ... 220, 301, 316, 502
fnot/1 ... 605, 607
foldl/4 ... 517, 518, 777, 782
footer/1 ... 865, 869
force_lazy/1 ... 43, 46
foreign/1 ... 661, 662
foreign/2 ... 661, 662
foreign_compilation ... 667, 669
foreign_inline/2 ... 665
foreign_interface ... 645
foreign_interface/foreign_interface_properties ... 465, 473, 546, 807
foreign_interface_properties ... 661
foreign_low/1 ... 661, 662
foreign_low/2 ... 661, 662
ForEmacs.txt ... 1112
form_assignment/1 ... 715, 719
form_default/3 ... 703, 706
form_dict/1 ... 715, 719
form_empty_value/1 ... 703, 706
form_request_method/1 ... 703, 710
form_value/1 ... 715, 719
format ... 59, 211, 218, 277, 388, 405, 455, 487, 685, 697, 763, 935, 961, 987
format/2 ... 218, 277
format/3 ... 218, 277, 278
format_control/1 ... 218, 277, 279
format_to_string(Format, Arguments, String) ... 278
format_to_string/3 ... 218, 277, 278, 281
formatting commands ... 339
formatting conventions, for emacs ... 67
formatting/2 ... 685, 686
formulae ... 435
forward/2 ... 919, 922
Francisco Bueno ... 9, 339, 355, 359, 413, 427, 455, 479, 511, 539, 589, 605, 685, 733, 735, 787, 793, 797, 803, 813, 829
free variable ... 475
freeze ... 579
freeze/2 ... 203, 579
freeze/freeze ... 995
frozen/2 ... 579
fsyntax ... 481, 482, 483, 525, 527
fun_eval/1 ... 525
fun_return/1 ... 525
func/1 ... 352
Function applications ... 525
functional ... 527, 529
Functional definitions ... 526
functional syntax ... 8, 507
functions ... 7
functor/3 ... 129, 131, 971
fuzzy ... 605
fuzzy/1 ... 605, 606
fuzzy_discrete/1 ... 605
fuzzy_predicate/1 ... 605, 606
fuzzybody/1 ... 605, 608

G

g_assrt_body/1 ... 349, 353
garbage collection ... 324, 326
garbage_collect/0 ... 224, 323, 327
garbage_collection_option/1 ... 225, 323, 326
gc/0 ... 182, 185
gc_result/1 ... 225, 323, 326
gcc ... 1109
gcd/2 ... 157
gen_assoc/3 ... 777, 778
genbar1 ... 852, 865
genbar2 ... 852, 871
genbar3 ... 852, 853, 875
genbar4 ... 853, 879
gendot ... 987
gendot/3 ... 987
generate_benchmark_list/7 ... 1083, 1085
generate_benchmark_list2/7 ... 1083, 1086
generate_human_file/0 ... 388, 389
generate_js_menu/1 ... 675, 678
generate_machine_file/0 ... 388, 389
generate_menu_path/2 ... 680
generate_plot/2 ... 989, 990, 1087

generate_plot/3 ..... 989, 990, 1087
generator 1021
generator/2 1021
generator_util 1023
gengraph1 ..... 854, 883
gengraph2 ..... 854, 855, 893
genmultibar 853, 901
geom_average/2 ..... 388, 390
Gerda Janssens 10
German Puebla 9, 339
get_active_config/1 ..... 487, 491
get_address/2 ..... 222, 301, 308, 504
get_alias_path/0 65
get_all_values/2 ..... 487, 488
get_arch/1 205
get_assoc/3 ..... 777, 779
get_assoc/5 ..... 777, 779
get_attribute/2 ..... 201, 203
get_byte/1 ..... 217, 259, 261
get_byte/2 ..... 217, 259, 261
get_char/1 ..... 217, 259, 262
get_char/2 ..... 216, 259, 262
get_ciao_ext/1 ..... 205, 206
get_code/1 169
get_code/2 169
get_cookies/1 ..... 703, 707
get_debug/1 ..... 205, 206
get_debugger_state/1 60
get_definition_dictionary/2 1009
get_dictionaries/2 ..... 1049, 1058
get_eng_location/1 ..... 205, 206
get_environment/2 ..... 1049, 1058
get_environment_name/2 ..... 1049, 1054
get_environment_type/2 ..... 1049, 1055
get_exec_ext/1 ..... 205, 206
get_first_parsed/3 ..... 1049, 1061
get_form_input/1 ..... 12, 703, 706, 711
get_form_value/3 ..... 703, 706
get_general_options/1 ..... 989, 1087
get_gid/1 ..... 223, 301, 307, 504
get_global/2 537
get_grnam/1 ..... 222, 301, 307, 504
get_indentation/2 ..... 1049, 1056
get_line/1 401
get_line/2 401
get_menu_configs/1 ..... 675, 676
get_menu_flag/3 ..... 675, 676
get_menu_flags/1 ..... 675, 677
get_menu_flags/2 ..... 675, 678
get_menu_options/2 ..... 675, 677
get_name/2 ..... 487, 492
get_name_value/3 ..... 487, 489
get_name_value_string/3 ..... 487, 489
get_next_assoc/4 ..... 777, 780
get_os/1 205
get_parsed/2 ..... 1049, 1057
get_perms/2 ..... 493, 499
get_pid/1 ..... 223, 301, 306, 505
get_platform/1 ..... 205, 206
get_prev_assoc/4 ..... 777, 780
get_primes/2 789
get_prototype_definition/2 ..... 1041, 1042
get_prototype_dictionary/2 ..... 1009, 1010
get_prototype_interface/2 1041
get_pwnam/1 ..... 223, 301, 307, 504
get_row_number/2 ..... 1049, 1055
get_settings_nvalue/1 ..... 487, 491
get_so_ext/1 ..... 205, 207
get_stream/2 411
get_tmp_dir/1 ..... 222, 301, 307, 504
get_type/2 739
get_uid/1 ..... 223, 301, 307, 504
get_value/2 ..... 487, 488
get_value_def/3 ..... 487, 488
get1_code/1 ..... 169, 170
get1_code/2 ..... 169, 170
getcounter/2 785
getct/2 ..... 169, 173
getct1/2 ..... 169, 174
getenvstr/2 ..... 224, 301, 304, 505
getopts 388, 391
getopts/1 392
getopts/4 391
GetStartUnix 21
GetStartWin32 27
ghostview 24, 29, 482
glb/2 ..... 982, 985
global 537
global variables 8, 507
GlobalChangeLog 77
gmake 833
gmax/3 581
gnd/1 ..... 105, 109
gndstr/1 ..... 105, 109
GNU 1101
GNU emacs 7, 31, 87

GNU general public license 74
GNU Library General Public License (LGPL) 1
gnuplot 989, 1087
gnuplot/gnuplot 1083
go/1 ..... 373, 374, 546, 549
go/2 ..... 373, 375, 546, 550
Goal 582
goal_available/1 ..... 546, 561
goal_cancelled/1 ..... 546, 565
goal_det/1 ..... 546, 562
goal_failed/1 ..... 546, 564
goal_finished/1 ..... 546, 563
goal_id/1 ..... 465, 468
goal_not_executed/1 ..... 546, 563
goal_rem_executing/1 ..... 546, 563
goal_tobacktrack/1 ..... 546, 564
goal_toreexecute/1 ..... 546, 564
Gopal Gupta 10
Goran Smedback .. 1001, 1005, 1007, 1009, 1013, 1015, 1017, 1019, 1021, 1023, 1035, 1039, 1041, 1047, 1049, 1063, 1065
grammar rule 403
granularity control 73
graph_b1/13 ..... 854, 884, 885, 886
graph_b1/9 ..... 854, 884, 885
graph_b2/13 ..... 854, 893, 894, 896
graph_b2/9 ..... 854, 893, 895
graph_w1/13 ..... 854, 884, 886
graph_w1/9 ..... 854, 884, 885
graph_w2/13 ..... 855, 893, 896
graph_w2/9 ..... 855, 893, 895
graphs 793
graphs/lgraphs 793
graphs/ugraphs 685, 793, 825
ground/1 ..... 121, 126, 359, 360, 362
guard/1 673
gunzip 1101, 1102

H

H. Ait-Kaci 42
halt/0 ..... 177, 179
halt/1 ..... 177, 179
halt_server/0 ..... 771, 772
handle_error/2 419
handler_type/1 860
hard side-effects 365
hash_power/1 ..... 957, 958, 975
hash_size/1 957
hash_term/2 511
have_choicepoints/1 ..... 359, 361
head pattern 349, 350, 353
head_pattern/1 ..... 349, 350, 353
header/1 ..... 865, 869
hello 63
help 24, 27, 29, 56, 1112
help, unix 22
help, windows 28
higher-order 8, 507
highest_hash_number/1 ..... 957, 958
hiord_rt 515
hiordlib 388, 517, 532, 777, 1083
hms_time/1 ..... 715, 721
hook_menu_check_flag_value/3 ..... 673, 675, 683
hook_menu_default_option/3 ..... 674, 675, 684
hook_menu_flag_help/3 ..... 673, 675, 684
hook_menu_flag_values/3 ..... 673, 675, 683, 684
hostname_address/2 ..... 473, 476
html 703
HTML 701, 703
html_expansion/2 ..... 703, 711
html_protect/1 ..... 703, 710
html_report_error/1 ..... 703, 706
html_template/3 ..... 703, 704
html_term/1 ..... 703, 715, 717
html2terms/2 703
http 713
HTTP 701, 713
http_date/1 ..... 715, 720, 721
http_lines/3 ..... 703, 710
http_request_param/1 ..... 715, 720
http_response_param/1 ..... 715, 720
hw 23
hw.pls 23, 29

I

icon_address.pl 701
icon_address/2 ..... 703, 710, 717
id 594, 595
identifier of a location 729, 737
idlists 447, 787, 813
if/3 ..... 99, 100
image/1 ..... 905, 906
IMDEA Software Institute 9
impl_defined/1 103

implements/1 ... 617, 619, 620, 624, 641
import/2 ... 93
imports ... 821
imports_meta_pred/3 ... 461
in/1 ... 373, 374, 546, 549, 771
in/2 ... 373, 375, 546, 549, 771, 982
in_circle_oc/3 ... 957, 959
in_circle_oo/3 ... 957, 959
in_noblock/1 ... 771
in_stream/2 ... 771, 772
inc_indentation/2 ... 1049, 1056
inccounter/2 ... 785
include/1 ... 43, 44, 97
incr_num_local_backtr/0 ... 546, 569
indentation_list/2 ... 1023, 1033
indep/1 ... 359, 360, 361, 362
indep/2 ... 359, 360, 361, 362
index/1 ... 512
indexer ... 511
indexer(hash) ... 511
indexer/hash ... 511, 957
indexspecs/1 ... 512
Inference of properties ... 73
info ... 21, 22, 67, 68, 70, 1102, 1104, 1105
INFOPATH ... 1104
inform_user/1 ... 197, 199
inherit_class/1 ... 617, 619
inheritable interface ... 618
inheritable/1 ... 617, 618
inheritance relationship ... 620
inherited/1 ... 617, 621
initial/0 ... 546, 550
initialization clauses ... 619
initialization file ... 24, 29
initialization/1 ... 104, 525
initialize_db/0 ... 725, 727
inner ... 377
INRIA ... 10
insert/3 ... 809, 919, 920
insert_after/3 ... 919, 921
insert_begin/3 ... 919, 921
insert_comments_in_beginning/3 ... 1049, 1054
insert_end/3 ... 919, 921
insert_last/3 ... 283, 291
insert_parsed/3 ... 1049, 1060
insert_top/3 ... 919, 920
inside_proto/1 ... 1049, 1058
inst/2 ... 105, 116
Install ... 1101
installation ... 9, 1099
installation, checking the ... 1105
installation, Mac OS X, full instructions ... 1102
installation, Mac OS X, summary ... 1101
installation, network based ... 1103
installation, Un*x, full instructions ... 1102
installation, Un*x, summary ... 1101
installation, Windows clients ... 1113
installation, Windows server ... 1113
installation, Windows, from binaries ... 1111
installation, Windows, from sources ... 1107
InstallWin32bin ... 1111
instance/2 ... 369, 445
instance_codes/2 ... 635, 637
instance_id/1 ... 635, 639
instance_of/2 ... 629, 630, 631, 632, 635, 636, 637
instances ... 635
instantiation mode ... 8, 337
instantiation properties ... 355
instantiation state ... 6
int/1 ... 105, 106, 156, 181
int_list/1 ... 661
integer/1 ... 121, 123, 157, 351
inter-process communication ... 585
intercept/3 ... 177, 178
interface ... 641
interface file ... 194
interface inheritance ... 620
interface/2 ... 635, 637
interface_name/1 ... 635, 639
interface_source/1 ... 635, 639
interfaces ... 617
internal_module_id/1 ... 205, 208
internal_types ... 1035
interp_file/2 ... 857, 858
interpreting ... 69, 70
intersect_vars/3 ... 447
intersection/3 ... 283, 292
intexpression/1 ... 151, 158
intlist/1 ... 789
intset_delete/3 ... 283, 292
intset_in/2 ... 283, 292
intset_insert/3 ... 283, 292
intset_sequence/3 ... 283, 292
io_alias_redirection ... 411, 1077
io_aux ... 181, 197
io_basic ... 169

io_mode/1 ..... 159, 167
IPL-D 10
is/2 ..... 151, 525, 535
is_array/1 775
is_assoc/1 ..... 777, 778
is_connected_to_java/0 ..... 763, 765
is_det/1 ..... 359, 361
is_dictionaries/1 1009
Isabel Martin Garcia .. 847, 857, 859, 861, 865, 871, 875, 879, 883, 893, 901, 905, 909, 911, 913, 915
iso 8, 229, 231
ISO-Prolog ... 4, 6, 156, 157, 169, 173, 193, 231, 255
ISO-Prolog builtins 8, 89, 229
iso-prolog, compliance 4
iso/1 ..... 105, 116
iso_byte_char 211, 216, 217, 231, 259, 383, 752, 1005, 1007, 1009, 1021, 1039, 1041, 1047, 1049
iso_incomplete 231, 267, 383, 1005, 1007, 1009, 1021, 1039, 1041, 1047, 1049
iso_misc 211, 217, 218, 231, 239, 265, 383, 1005, 1007, 1009, 1021, 1039, 1041, 1047, 1049
isomodes 6, 371
issue_debug_messages/1 ..... 405, 408, 409, 410
iterative deepening-based execution 9, 507
iterative-deepening 595

J

Jan Maluzynski 10
Java event handling from Prolog 750
Java exception handling from Prolog 752
Java interface 9, 643
Java to Prolog interface 759
java_add_listener/3 ..... 752, 756
java_connect/2 ..... 752, 753
java_constructor/1 ..... 752, 754
java_create_object/2 ..... 752, 754
java_debug/1 ..... 763, 765
java_debug_redo/1 ..... 763, 765
java_delete_object/1 ..... 752, 755
java_disconnect/0 ..... 752, 753
java_event/1 ..... 752, 754
java_field/1 ..... 752, 754
java_get_value/2 ..... 752, 756
java_invoke_method/2 ..... 752, 755
java_method/1 ..... 752, 755
java_object/1 ..... 752, 754
java_query/2 ..... 763, 764
java_remove_listener/3 ..... 752, 757
java_response/2 ..... 763, 764
java_set_value/2 ..... 752, 756
java_start/0 752
java_start/1 752
java_start/2 ..... 752, 753
java_stop/0 ..... 752, 753
java_use_module/1 ..... 752, 754
javall/javasock 752, 759
javall/jtopl 763
javart 749, 763
javasock 763
Jesus Correas 747, 749, 759, 763
Johan Andersson 87
Johan Bevemyr 87
Johan Widen 10
John Gallagher 10
join_socket_interface/0 ..... 763, 764
Jose F. Morales ... 205, 209, 323, 521, 525, 571, 575, 645, 661, 667, 669
Jose Manuel Gomez Perez .. 433, 723, 981, 1089, 1093
jtopl 759, 763
just_benchmarks/0 ..... 388, 389, 390

K

K. Shen 49, 59
K.U. Leuven 10
Kalyan Muthukumar 10
Kevin Greene 10
key sequences 68
keyboard 6
keylist/1 ..... 218, 295, 296
keypair/1 ..... 218, 295, 296
keysort/2 ..... 218, 295, 296
keyword/1 ..... 730, 737
Kim Marriott 10

L

L. Byrd 10
L.M. Pereira 10
labeling/1 ..... 982, 983
last/2 ..... 212, 283, 291
latex 831
lazy 525, 993
lazy/1 995
leap 54

leash/1 ..... 54, 60
Lena Flood 775, 809
length/2 ..... 212, 283, 288, 919, 922
length_next/2 ..... 919, 923
length_prev/2 ..... 919, 923
letter_match/2 ..... 791, 792
lgraph/1 687
lgraph/2 803
lgraphs 803
lib library 8, 89
libpaths 43, 65
libraries used 821
library directory 1103
library(actmods/examples/webbased_ server/webbased_server) 590
library(basicmodes) 350
library(iso_byte_char) 169
library(isomodes) 350
library(pure) 91
library/pillow/doc 701
library_directory/1 ..... 34, 65, 66, 159, 167
librowser 457
limitations 9, 1099
limitations, architecture-specific 1115
linda 771
linda_client/1 771
linda_timeout/2 ..... 771, 772
line/1 401
line_count/2 ..... 159, 162
line_position/2 ..... 159, 163
linear/1 ..... 359, 362
linker_and_opts/2 667
Linkoping U. 10
Linux 1106
list/1 ..... 105, 111, 116
list/2 ..... 105, 112, 351
list_breakpt/0 60
list_concat/2 ..... 283, 291
list_functor/1 ..... 129, 134
list_insert/2 ..... 283, 291, 787
list_lookup/3 ..... 283, 292
list_lookup/4 ..... 283, 292
list_of_lists/1 ..... 283, 294
list_to_assoc/2 ..... 777, 780
list_to_conj/2 435
list_to_conj/3 435
list_to_disj/2 ..... 435, 436
list_to_disj2/2 ..... 435, 439
list_to_list_of_lists/2 ..... 283, 294
list1/2 ..... 283, 290
lists 65, 211, 212, 233, 245, 283, 301, 340, 343, 359, 388, 391, 395, 405, 423, 449, 457, 479, 487, 493, 669, 675, 703, 713, 725, 745, 752, 768, 777, 789, 791, 865, 871, 875, 879, 884, 893, 901, 905, 909, 911, 913, 915, 919, 989, 1009, 1017, 1039, 1047, 1049, 1065, 1073, 1077, 1083, 1089
literal 378
llist_to_conj/2 ..... 435, 437
llist_to_disj/2 ..... 435, 438
llists 395, 493, 669
load_compilation_module/1 ..... 194, 195
loading mode 50
loading programs 22, 28, 69
loading_code 97
locating errors 72
location_t/1 410
lock_atom/1 ..... 465, 468
lock_file/3 433
log of changes 74
log/1 157
LogIn 42
long 378
look_ahead/3 ..... 1049, 1062
look_first_parsed/2 ..... 1049, 1061
lookup 1041
lookup_check_field/6 ..... 1041, 1042
lookup_check_interface_fieldValue/8 ..... 1041, 1043
lookup_check_node/4 ..... 1041, 1042
lookup_field/4 ..... 1041, 1043
lookup_field_access/4 ..... 1041, 1044
lookup_fieldTypeId/1 ..... 1041, 1044
lookup_get_fieldType/4 ..... 1041, 1044
lookup_route/5 ..... 1041, 1044
lookup_set_def/3 ..... 1041, 1045
lookup_set_extern_prototype/4 ..... 1041, 1046
lookup_set_prototype/4 ..... 1041, 1045
lpdoc 1, 3, 7, 31, 67, 68, 74, 75, 79, 80, 339, 346, 350, 354
LPdoc 3
lpdoc command args, setting 78
lpdoc command, setting 78
lpdoc default format, setting 77
lpdoc lib path, setting 78
lpdoc working dir, setting 78
lpmake 481, 483, 487, 831

lpmake autodocumentation 831
ls/2 ..... 493, 495
ls/3 ..... 493, 495
lub/2 ..... 982, 985

M

machine_name/1 ..... 752, 753
mailing list 9, 1099, 1115
main module 70
main/0 ..... 23, 28, 29, 33, 34, 69
main/1 ..... 23, 25, 28, 29, 30, 33, 34, 51, 63, 64, 69, 181, 388, 389, 391, 935
major version number 74
make 481, 483, 485, 487, 831, 1101
make/1 487
make/make_rt 481
make/up_to_date 487
make_actmod/2 ..... 43, 46, 586
make_directory/1 ..... 222, 301, 309, 504
make_directory/2 ..... 222, 301, 309, 504
make_dirpath/1 ..... 222, 301, 309, 503
make_dirpath/2 ..... 222, 301, 309, 504
make_exec/2 ..... 23, 28, 43, 44
make_option/1 ..... 487, 488
make_persistent/2 ..... 724, 725, 728
make_po/1 ..... 43, 45, 297
make_rt 487
make_wam/1 297
Makefile 33, 481, 483, 491, 493
Makefile.pl 482, 831
man 1104
MANPATH 1104
manual, printing 24, 27, 29, 1112
manual, tour 7
manuals 1103
manuals, printing 24, 29
Manuel C. Rodriguez 49, 59, 67
Manuel Carro ... 9, 187, 201, 205, 245, 249, 255, 301, 323, 385, 391, 411, 433, 465, 471, 473, 517, 571, 575, 577, 579, 581, 593, 595, 645, 661, 667, 669, 777, 797, 819, 835, 837, 839, 841, 981, 1001, 1101, 1111, 1115
Manuel Hermenegildo .. 9, 10, 21, 27, 63, 67, 99, 105, 121, 129, 135, 151, 233, 249, 259, 339, 349, 355, 359, 371, 373, 413, 481, 487, 493, 521, 525, 585, 593, 595, 701, 703, 723, 821, 827, 831, 1101, 1111, 1115
map/3 ..... 517, 777, 781
map/4 ..... 517, 518
map_assoc/2 ..... 777, 781
map_assoc/3 ..... 777, 781
Maria Jose Garcia de la Banda 9
marshalling 8, 381
Masanobu Umeda 87
match_pattern/2 791
match_pattern/3 791
match_pattern_pred/2 ..... 791, 792
match_posix/2 ..... 1070, 1073
match_posix/3 ..... 1070, 1074
match_posix/4 ..... 1070, 1073, 1074
match_posix_matches/3 ..... 1070, 1073, 1074
match_posix_rest/3 ..... 1070, 1073, 1074
match_pred/2 ..... 1071, 1073, 1075
match_shell/2 ..... 1070, 1073
match_shell/3 ..... 1069, 1070, 1073
match_struct/4 ..... 1071, 1073, 1075
Mats Carlsson 10, 49, 59, 87, 159, 169, 181, 233, 249, 323, 797
Maurice Bruynooghe 10
max/3 581
max_assoc/3 ..... 777, 778
maxdepth/1 60
MCC 10
Melbourne U. 10
member 681
member/2 ..... 105, 113
member_0/2 787
member_var/2 447
memberchk/2 787
memo/1 ..... 105, 119
memory management 324, 326
memory_option/1 ..... 225, 323, 326
memory_result/1 ..... 225, 323, 326
menu 673
Menu 682
menu/1 675
menu/2 675
menu/3 675
menu/4 675
menu/menu_generator 673
menu/menu_rt 673
menu_default/3 ..... 673, 675, 682, 684
menu_flag_values/1 ..... 675, 681
menu_generator 675
menu_opt/6 ..... 673, 675, 679, 681, 682

merge/3 ..... 809, 812
merge_tree/2 ..... 1009, 1011
MERIT 10
message/1 ..... 197, 198, 199
message/2 ..... 197, 198, 200
message_info/1 ..... 197, 199
message_lns/4 ..... 197, 198
message_t/1 ..... 405, 410
message_type/1 ..... 197, 200
messages 405, 435, 455, 487, 493, 669, 675
messages/1 ..... 197, 198, 200
meta_predicate/1 ..... 94, 617
meta_predname/1 ..... 725, 728
metaprops/meta_props 373
metaspec/1 94
method_spec/1 ..... 635, 639
mfclause/2 ..... 239, 242
mfstringValue/5 1017
mfstringValue/7 ..... 1019, 1020
MICYT 10
min_assoc/3 ..... 777, 778
minimum/3 ..... 517, 518
minor version number 74
mkdir_perm/2 ..... 493, 499
mkf-CIAOARCH 1108
mktemp/2 ..... 220, 301, 314, 502
mktemp_in_tmp/2 ..... 220, 301, 314, 502
MOBIUS 10
mod/2 157
mode 6
mode 340, 350
mode spec 6
mode_of_module/2 ..... 297, 298
modedef/1 ..... 6, 340, 345, 350
modes 73, 371, 373
modif_time/2 ..... 220, 301, 316, 502
modif_time0/2 ..... 220, 301, 316, 502
modular interface 37
module qualification 91
module/2 ..... 92, 211
module/3 ..... 51, 91, 92, 97, 617, 641
module_address/2 ..... 589, 590
module_of/2 ..... 297, 298
modulename/1 94
modules 91
modules, active 37
Monash U. 10
month/1 ..... 715, 721
Montse Iglesias Urraca 689, 697
more_solutions/1 ..... 546, 566
most_general_instance/3 ..... 13, 445
most_specific_generalization/3 ..... 13, 445
move_execution_top/1 ..... 546, 566
move_file/2 ..... 493, 494
move_files/2 493
moving changelog entries 76
mshare/1 ..... 359, 362
multi-evaluated 618
multiarchitecture support 1106
multibar_attribute/1 903
multibarchart/10 ..... 853, 901, 902
multibarchart/8 ..... 853, 901, 902
multifile predicate 103
multifile/1 ..... 43, 47, 91, 103, 617
multifile:alias_file/1 427
mut_exclusive/1 ..... 359, 362
my_url/1 ..... 703, 708
mycin 997

N

n = arity 391
n_assrt_body/5 ..... 352, 353
nabody/1 ..... 349, 352
Name 662, 663
name server 586
name/2 ..... 141, 706
name_value/2 ..... 487, 489
Naming term arguments 521
native/1 ..... 105, 118
native/1,2 12
native/2 ..... 105, 118
native_props 359
nativeprops.pl 377
needs_state/1 ..... 661, 663
neighbors/3 797
neq/3 ..... 675, 681
netscape 21, 1105
New Mexico State University 10
new/2 ..... 621, 629, 630, 631, 635, 636, 637
new_array/1 775
new_atom/1 ..... 225, 323, 325
new_declaration/1 ..... 103, 193
new_declaration/2 ..... 103, 193
new_interp/1 ..... 697, 857
new_interp/2 697

new_interp_file/2 ... 697, 698
new_measure/0 ... 546, 569
newer/2 ... 487, 490
next/2 ... 919, 920
next_on_circle/2 ... 957, 958
nl/0 ... 169, 172
nl/1 ... 169, 172
nlist/1 ... 112
nlist/2 ... 105, 112
nnegint/1 ... 105, 106
no ... 377, 378
no_choicepoints/1 ... 359, 362
no_exception/1 ... 359, 363
no_exception/2 ... 359, 363
no_path_file_name/2 ... 423
no_rtcheck/1 ... 105, 118
no_signal/1 ... 359, 363
no_signal/2 ... 359, 363
no_swapslash/3 ... 219, 301, 320, 500
no_tr_nl/2 ... 493, 498
nobreakall/0 ... 60
nobreakpt/6 ... 50, 51, 60
nocontainsx/2 ... 283, 291
node_id/2 ... 961, 962, 963, 964, 965, 966, 967, 968, 969
nodebug ... 50, 55
nodebug/0 ... 60
nodebug_module/1 ... 50, 60
nodebugrtc/0 ... 61
nodeDeclaration/4 ... 1021, 1047
nofileerrors/0 ... 182, 185
nogc/0 ... 182, 185
non-failure ... 73
non_det/1 ... 359, 363
non_empty_dictionary/1 ... 399, 400
non_empty_list/1 ... 129, 132
nonground/1 ... 12, 359, 363
nonsingle/1 ... 283
nonvar/1 ... 121, 122
normal_message/2 ... 487, 490
nospy ... 56
nospy/1 ... 50, 51, 57, 61
nospyall/0 ... 61
not_covered/1 ... 359, 363
not_empty/3 ... 915, 916
not_empty/4 ... 915
not_fails ... 379
not_fails/1 ... 359, 364
not_further_inst/1 ... 352
not_further_inst/2 ... 105, 117
not_in_circle_oc/3 ... 957, 958
not_measure/0 ... 546, 569
not_mut_exclusive/1 ... 359, 364
notation ... 5
note/1 ... 197, 198, 199
note_message/1 ... 405, 407
note_message/2 ... 405, 407
note_message/3 ... 405, 407
notrace/0 ... 61
ntemacs ... 1105, 1112
nth/3 ... 212, 283, 289
null/1 ... 661
null_ddlist/1 ... 919
null_dict/1 ... 813
num/1 ... 105, 107
num_code/1 ... 105, 115
num_solutions/2 ... 359, 364
number/1 ... 121, 125
number_agents/1 ... 546, 560
number_chars/2 ... 217, 259, 260
number_codes/2 ... 141, 143
number_codes/3 ... 141
numbervars/3 ... 215, 249, 253
numlist/1 ... 789
numlists ... 789

O

object ... 635
object oriented programming ... 9, 507
objects ... 629
objects/objects_rt ... 617, 629
objects_rt ... 635
ociao ... 611
odd ... 577
old_database ... 211, 226, 227, 331
old_or_new/1 ... 399, 400
on-line help ... 67
on_abort/1 ... 104, 525
once/1 ... 218, 265
op/3 ... 193, 211, 255
open/3 ... 13, 159, 427
open/4 ... 159, 160
open_client/2 ... 771, 772
open_DEF/5 ... 1023, 1031
open_EXTERNPROTO/5 ... 1023, 1031

open_input/2 ... 397
open_node/6 ... 1023, 1029
open_null_stream/1 ... 397
open_option_list/1 ... 159, 160
open_output/2 ... 397
open_predicate/1 ... 187, 191
open_PROTO/4 ... 1023, 1030
open_Script/5 ... 1023, 1032
operations file ... 724
operator table ... 193
operator_specifier/1 ... 105, 110
operators ... 211, 231, 245, 249, 255, 335, 383, 451, 745, 1005, 1007, 1009, 1021, 1039, 1041, 1047, 1049
optional_message/2 ... 405, 408
optional_message/3 ... 405, 408
ord_delete/3 ... 809
ord_disjoint/2 ... 809, 812
ord_intersect/2 ... 809, 810
ord_intersection/3 ... 809, 810
ord_intersection_diff/4 ... 809, 810
ord_list_to_assoc/2 ... 777, 781
ord_member/2 ... 809
ord_subset/2 ... 809, 811
ord_subset_diff/3 ... 809, 811
ord_subtract/3 ... 809, 810
ord_test_member/3 ... 809, 810
ord_union/3 ... 809, 811
ord_union_change/3 ... 809, 812
ord_union_diff/4 ... 809, 811
ord_union_symdiff/4 ... 809, 812
Oscar Portela Arjona ... 421
Other functionality ... 527
otherwise/0 ... 99, 102
out/1 ... 373, 374, 546, 549, 771, 1039
out/2 ... 373, 375, 546, 549
out/3 ... 1039
out_stream/2 ... 771, 772
output_error/1 ... 1013
output_html/1 ... 703, 711
output_to_file/2 ... 429, 431
overriden ... 618, 619

P

P. Lincoln ... 42
Pablo Chico ... 777
package file ... 92, 97, 193, 194, 195
package/1 ... 92
pair/1 ... 795
parallel programming ... 8, 507
parallel Prolog ... 10
parallelizing compiler ... 10
parametric type functor ... 358
PARFORCE ... 10
parse/1 ... 1017, 1018, 1035, 1036
parse_term/3 ... 413, 414
parse_term/4 ... 413, 415
parser/2 ... 1047
parser_util ... 1049
partial evaluation ... 73
passerta_fact/1 ... 725, 726, 727
passertz_fact/1 ... 725, 726, 727
patch number ... 75
PATH ... 1103
path alias ... 46, 65, 92, 165, 167, 301
path aliases ... 35
path/1 ... 539, 541
pattern/1 ... 493, 500, 791, 792, 861, 863
pattern/2 ... 861, 863
patterns ... 459, 791, 1069
Paulo Moura ... 13
pause/1 ... 224, 301, 506
Pawel Pietrzak ... 10
pe_type/1 ... 105, 119
Pedro Lopez ... 9, 355, 359
peek_byte/1 ... 217, 259, 261
peek_byte/2 ... 217, 259, 261
peek_char/1 ... 216, 259, 263
peek_char/2 ... 216, 259, 263
peek_code/1 ... 169, 171, 263
peek_code/2 ... 169, 170, 263
PEPMA ... 10
percentbarchart1/7 ... 852, 865, 867
percentbarchart2/7 ... 852, 871, 873
percentbarchart3/7 ... 853, 875, 876
percentbarchart4/7 ... 853, 879, 880
performance/3 ... 1083
perl ... 85
persdb ... 724, 725, 727, 733, 737
persdb/persdbcache ... 725, 735
persdb/persdbrt ... 675
persdb_sql ... 723, 724
persdbrt ... 723, 737
persdbtr_sql ... 743
persistence set ... 724

persistent 726, 728
persistent predicate 723
Persistent predicate 9, 643
persistent/2 ..... 724, 729
persistent_dir 729
persistent_dir/2 ..... 675, 681, 725, 727, 729, 730, 735, 736, 737
persistent_dir/2-4 724
persistent_dir/4 ..... 13, 675, 682, 725, 729
Peter Olin 87
Peter Stuckey 10
phrase/2 275
phrase/3 275
Pierre Deransart 10
pillow 701, 724, 1113
PiLLoW on-line tutorial 701
pillow.pl 701
pillow/html 701, 1001, 1089
pillow/http 701, 1001, 1089
pillow/http_ll 713
pillow/pillow_aux 703, 713
pillow/pillow_types 703, 713, 1089
pillow_types 715
pipe/2 ..... 159, 165
pitm/2 ..... 982, 983
pkunzip 1111
pl2sqlinsert 745
pl2sqlInsert/2 745
Platform independence 747
platform-dependent 36
platform-independent 35, 36
point_to/3 ..... 797, 799
Polymorphism 611
pop_active_config/0 ..... 487, 491
pop_ciao_flag/1 ..... 182, 184
pop_global/2 537
pop_name_value/1 ..... 487, 491
pop_prolog_flag/1 ..... 182, 183
popen/2 311
popen/3 ..... 221, 301, 311, 503
popen_mode/1 ..... 221, 301, 311, 503
portray/1 ..... 249, 250, 254
portray_attribute/2 ..... 203, 249, 250, 254
portray_clause/1 ..... 215, 249, 253
portray_clause/2 ..... 215, 249, 252, 253
Posix threads 1108
posix_regexp/1 ..... 1073, 1075
possible 1063
possibly_fails/1 ..... 359, 365
possibly_nondet/1 ..... 359, 365
postgres2sqltype/2 ..... 739, 741
postgres2sqltypes_list/2 ..... 739, 741
postgrestype/1 ..... 739, 741
powerset/2 ..... 283, 294
pred assertion 340, 341
pred/1 ..... 340, 341, 342, 343, 345, 349, 352
pred/2 ..... 340, 341
pred1/Arity1 999
Predefined evaluable functors 525
predfunctor/1 ..... 349, 354
predicate 378
predicate declarations 821
predicate spec 6
predicate spec 470
predicate/n 379
predicate_property/2 ..... 224, 323, 327, 328
predicate_property/3 ..... 224, 323, 327
predN/ArityN 999
predname/1 ..... 105, 115, 350
preprocessing programs 73
preprocessor 7, 31
preprocessor command args, setting 78
preprocessor command, setting 78
pretract_fact/1 ..... 725, 726, 727
pretractall_fact/1 ..... 12, 725, 726, 727
pretty_print 451
pretty_print/2 451
pretty_print/3 451
pretty_print/4 451
prettyvars/1 ..... 215, 249, 253, 815
prettyvars/2 ..... 813, 815
prev/2 ..... 919, 920
print 55
print/1 ..... 56, 215, 249, 252
print/2 ..... 215, 249, 252
print_measures/0 ..... 546, 569
printable_char/1 ..... 215, 249, 253
printdepth 56
printing assertion information 821
printing code-related information 821
printing, manual 24, 27, 29, 1112
Procedure Box 49
profiler 999
program assertions 339
program development environment 67
program development tools 1103

program parallelization 73
program specialization 73
program transformations 67, 73
programming environment 7, 31
project files 33
prolog flag 41, 42, 63, 103, 164, 181, 197, 1069
prolog flags 1069
Prolog server 759, 760
Prolog shell scripts 63
Prolog to Java Interface Structure 749
Prolog to Java Interface Structure. Java side 749
Prolog to Java Interface Structure. Prolog side 749
prolog-emacs interface 767
prolog.el 87
prolog_flag/3 182
prolog_flags 181
prolog_goal/1 ..... 752, 754
prolog_predicate/N 645
prolog_query/2 ..... 763, 764
prolog_response/2 ..... 763, 765
prolog_server/0 759
prolog_server/1 ..... 759, 760
prolog_server/2 ..... 759, 760
prolog_sys ... 211, 224, 225, 226, 239, 323, 388, 465, 471, 635, 1083
PrologName ... 468, 469, 473, 474, 475, 476, 477, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 662, 807, 808
PROMESAS 10
PROMETIDOS 10
prompt 675, 685
prompt/2 ..... 182, 184
prop assertion 343, 344
prop/1 ..... 340, 343, 344
prop/2 ..... 340, 344
properties of computations 355
properties of execution states 355
properties, basic 105
properties, native 359
property 343
property compatibility 116
property declarations 821
property_conjunction/1 ..... 349, 350, 351
property_starterm/1 ..... 349, 350, 351
propfunctor/1 ..... 349, 354
protected 618
protocol/1 589
providing information to the compiler ... 344, 347
provrml 1001
ProVRML 1001
provrml/boundary 1019, 1041
provrml/dictionary 1041
provrml/dictionary_tree 1041, 1049
provrml/field_type 1041
provrml/field_value 1023, 1041, 1047
provrml/field_value_check 1023, 1041
provrml/generator 1001, 1019
provrml/generator_util 1019, 1021, 1041
provrml/internal_types 1005, 1007, 1009, 1021, 1041, 1049
provrml/lookup 1023, 1047
provrml/parser_util 1017, 1019, 1021, 1023, 1041, 1047
provrml/possible 1047
provrml/provrml_io ... 1001, 1019, 1021, 1023, 1041
provrml/provrml_parser 1001, 1017
provrml/provrmlerror 1005, 1017, 1021, 1023, 1041, 1047, 1065
provrml/tokeniser 1019, 1047
provrml_io 1039
provrml_parser 1047
provrmlerror 1013
prune_dict/3 ..... 813, 814
public 619
public domain 1
public interface 618
public/1 ..... 617, 618, 625
pure 8, 89, 509
pure Prolog 8, 507, 509
push_active_config/1 ..... 487, 491
push_ciao_flag/2 ..... 182, 184
push_dictionaries/3 ..... 1049, 1057
push_global/2 537
push_goal/1 ..... 546, 561
push_goal/3 ..... 546, 560
push_name_value/2 491
push_name_value/3 ..... 487, 491
push_prolog_flag/2 ..... 182, 183
push_whitespace/3 ..... 1049, 1057
put_assoc/4 ..... 777, 782
put_assoc/5 ..... 777, 782, 783
put_byte/1 ..... 217, 259, 262
put_byte/2 ..... 217, 259, 262
put_char/1 ..... 216, 259, 263

put_char/2 ... 216, 259, 263
put_code/1 ... 169, 172, 263
put_code/2 ... 169, 171, 264
put_value/5 ... 783
putbyte/2 ... 262

Q

q_delete/3 ... 805
q_empty/1 ... 805
q_insert/3 ... 805
q_member/2 ... 805
query ... 41
query_requests/2 ... 759, 760
query_solutions/2 ... 759, 760
queues ... 805
quoted string ... 157
Quoting functors ... 526

R

random ... 807
random/1 ... 807
random/3 ... 807
random/random ... 861, 865, 871, 875, 879, 884, 893, 901
random_color/1 ... 861, 863
random_darkcolor/1 ... 861, 864
random_lightcolor/1 ... 861, 863
random_pattern/1 ... 861, 864
rd/1 ... 771, 772
rd/2 ... 771, 772
rd_findall/3 ... 771, 772
rd_noblock/1 ... 771, 772
reachability/4 ... 825
read ... 211, 214, 215, 231, 245, 383, 427, 429, 457, 493, 685, 697, 703, 725, 735, 752, 759, 771, 1005, 1007, 1009, 1021, 1039, 1041, 1047, 1049
read/1 ... 199, 215, 245
read/2 ... 214, 245, 246, 473, 474
read_event/1 ... 546, 566
read_option/1 ... 214, 245, 246
read_page/2 ... 1004
read_pr ... 979
read_pr/2 ... 979
read_term/[2,3] ... 246
read_term/2 ... 214, 245, 246, 251
read_term/3 ... 175, 214, 245, 246, 250
read_terms_file/2 ... 1039, 1040
read_top_level/3 ... 214, 245, 246
read_vrml_file/2 ... 1039, 1040
readf/2 ... 493, 497
reading/4 ... 1023
reading/5 ... 1023, 1025
reading/6 ... 1023, 1029
rebuild_foreign_interface/1 ... 669
rebuild_foreign_interface_explicit_decls/2 ... 669, 670
rebuild_foreign_interface_object/1 ... 669, 671
receive_confirm/2 ... 697, 699
receive_event/2 ... 697, 699
receive_list/2 ... 697, 699
receive_result/2 ... 697, 698
recorda/3 ... 227, 331
recorded/3 ... 226, 331, 332
records ... 8, 507, 521
recordz/3 ... 227, 331
recursive level ... 42
recycle_term/2 ... 449
redefined ... 621
redefining/1 ... 104
RedHat 5.0 ... 1109
reduce_indentation/3 ... 1049, 1057
reexport/1 ... 94
reexport/2 ... 93
reference/1 ... 187, 192
references, to Ciao ... 5
referring to Ciao ... 5
regedit ... 1114
regexp ... 1069
regexp/regexp_code ... 457, 493, 1069
regexp_code ... 1073
register_module/1 ... 487, 490
regtype assertion ... 358
regtype/1 ... 105, 118, 357, 358, 546, 547
regtype/2 ... 357, 358, 546, 547
regtypes ... 355
regular expresions ... 459
regular expressions ... 791
regular type ... 358
regular type definitions ... 355
regular type expression ... 358
regular types ... 355
relations/2 ... 359, 365
release/1 ... 546, 567
release_all_for_unwinding/0 ... 546, 567

release_remote/1 ..... 546, 567
release_some_suspended_thread/0 ..... 546, 567
rem/2 157
remove_all_elements/3 ..... 919, 922
remove_code/3 ..... 1049, 1062
remove_comments/4 ..... 1023, 1034
remove_menu_config/1 ..... 675, 677
Remy Haemmerle 579, 595, 845
rename/2 ..... 813, 815
rename_file/2 ..... 219, 301, 318, 493, 494, 501
repeat/0 ..... 99, 101
replace_all/4 ..... 1071, 1073, 1075
replace_characters/4 ..... 219, 301, 320, 500
replace_first/4 ..... 1071, 1073, 1075
replace_strings/3 ..... 493, 498
replace_strings_in_file/3 ..... 493, 498
reporting bugs 9, 1099, 1116
reserved_words/1 ..... 1005, 1006
restore_flags_list/1 677
restore_menu_config/1 ..... 675, 677
restore_menu_flags/2 ..... 675, 678
restore_menu_flags_list/1 ..... 675, 678
retract/1 ..... 213, 239, 241
retract_fact/1 ... 187, 189, 190, 723, 725, 726, 727, 735, 736
retract_fact_nb/1 ..... 187, 190
retractall/1 ..... 213, 239, 241
retractall_fact/1 ..... 12, 187, 190, 725, 727
retrieve_goal/2 ..... 546, 562
retrieve_list_of_values/2 ..... 982, 986
retrieve_range/2 ..... 982, 984
retrieve_store/2 ..... 982, 984
retry 55
returns/2 ..... 661, 662
reverse/2 ..... 212, 283, 286
reverse/3 ..... 283, 286
reverse_parsed/2 ..... 1049, 1061
rewind/2 ..... 919, 922
Richard A. O'Keefe 233, 249, 295, 471, 797
Robert Manchek 835
Roger Nasr 10, 42
rooted_subgraph/3 ..... 797, 799
round/1 157
row/1 906
rtchecks 377
rtchecks_asrloc 377
rtchecks_callloc 378
rtchecks_entry 377
rtchecks_exit 377
rtchecks_inline 377
rtchecks_level 377
rtchecks_namefmt 378
rtchecks_predloc 377
rtchecks_rt.pl 377
rtchecks_test 377
rtchecks_trust 377
run-time checks 343
run-time libraries 1103
run-time tests 73
run_tester/10 1077
running programs 22, 23, 28, 29
running unit tests 379
running_queries/2 ..... 759, 760
runtime_ops 335

S

s_assrt_body/1 ..... 349, 352
Sacha Varma 703
safe_write/2 479
Samir Genaim 597, 601
Saumya Debray 10
save_addr_actmod/1 ..... 589, 590
save_end_execution/1 ..... 546, 566
save_init_execution/1 ..... 546, 566
save_menu_config/1 ..... 675, 676, 677
scattergraph_b1/12 ..... 854, 884, 887
scattergraph_b1/8 ..... 854, 884, 887, 888
scattergraph_b2/12 ..... 855, 893, 897
scattergraph_b2/8 ..... 855, 893, 896
scattergraph_w1/12 ..... 854, 884, 889
scattergraph_w1/8 ..... 854, 884, 888, 898
scattergraph_w2/12 ..... 855, 893, 898
scattergraph_w2/13 898
scattergraph_w2/8 ..... 855, 893, 898
scattergraph1_b1/13 889
script header, inserting automatically 72
scripts 21, 23, 29, 1103, 1105
SCUBE 10
second_prompt/2 ..... 214, 245, 246
see/1 ..... 226, 329
seeing/1 ..... 226, 329
seen/0 ..... 226, 329
Seif Haridi 10
select/3 ..... 211, 283, 288
select_socket/5 ..... 473, 475

self/1 ... 590, 617, 621
semantic analisys ... 636
semaphore ... 468, 469
send_event/1 ... 546, 565
send_info_to_developers/0 ... 388, 389
send_signal/1 ... 177, 178, 179
send_silent_signal/1 ... 177, 178
send_term/2 ... 697, 699
sequence/2 ... 105, 113
sequence_or_list/2 ... 105, 113
sequence_to_list/2 ... 283, 294
Sergio Guadarrama ... 605
serve_socket/3 ... 479
set_ciao_flag/2 ... 182, 184
set_cookie/2 ... 703, 707
set_debug_mode/1 ... 43, 46, 50, 297, 298
set_debug_module/1 ... 297, 298
set_debug_module_source/1 ... 297, 298
set_env/2 ... 223, 301, 304, 505
set_environment/3 ... 1049, 1060
set_exec_mode/2 ... 220, 301, 317, 501
set_exec_perms/2 ... 493, 499
set_fact/1 ... 187, 191
set_general_options/1 ... 989, 1087
set_global/2 ... 537
set_goal_cancelled/1 ... 546, 565
set_goal_det/1 ... 546, 562
set_goal_failed/1 ... 546, 565
set_goal_finished/1 ... 546, 563
set_goal_nondet/1 ... 546, 562
set_goal_not_executed/1 ... 546, 563
set_goal_rem_executing/1 ... 546, 563
set_goal_tobacktrack/1 ... 546, 564
set_goal_toreexecute/1 ... 546, 564
set_input/1 ... 159, 161
set_menu_flag/3 ... 675, 676
set_name_value/2 ... 487, 489
set_nodebug_mode/1 ... 43, 46, 50, 297, 298
set_nodebug_module/1 ... 297, 298
set_output/1 ... 159, 161
set_owner/2 ... 493, 497
set_parsed/3 ... 1049, 1060
set_perms/2 ... 493, 499
set_prolog_flag/1 ... 186, 626
set_prolog_flag/2 ... 182, 183
set_stream/3 ... 411
setarg/3 ... 577
setcounter/2 ... 785
setenvstr/2 ... 224, 301, 304, 505
setof/3 ... 213, 233, 471, 472
setproduct/3 ... 809, 812
sets ... 582, 797, 801, 803, 809, 813, 825
SETTINGS ... 1105, 1108, 1109
sformat/3 ... 218, 277, 278
sh ... 22, 87, 1102, 1104
sh-compatible shell ... 22, 1102, 1104
sharing sets ... 362
shell ... 27
shell scripts ... 33
shell/0 ... 221, 301, 310, 503
shell/1 ... 221, 301, 310, 503
shell/2 ... 221, 301, 310, 503
shell/n ... 321
shell_regexp/1 ... 1073, 1075
shell_s/0 ... 759, 760
short ... 378
shortcut, windows ... 1111
show_handler/1 ... 546, 565
show_menu_config/1 ... 675, 677
show_menu_configs/0 ... 675, 677
show_message/2 ... 405, 409
show_message/3 ... 405, 409
show_message/4 ... 405, 409
shutdown_type/1 ... 473, 477
SICS ... 10, 87
SICStus ... 79
SICStus Prolog ... 10
sideff/2 ... 105, 117
sideff_hard/1 ... 359, 365
sideff_pure/1 ... 359, 365
sideff_soft/1 ... 359, 366
sign/1 ... 157
signal/1 ... 359, 366
signal/2 ... 359, 366
signals/2 ... 359, 366
simple_client.pl ... 586
simple_message/1 ... 405, 407
simple_message/2 ... 405, 407
sin/1 ... 157
site-specific programs ... 1103
size/1 ... 884, 891
size/2 ... 359, 366
size/3 ... 359, 366
size_lb/2 ... 359, 366
size_metric/3 ... 359, 367
size_metric/4 ... 359, 367

size_o/2 ..... 359, 366
size_of/2 646
size_of/3 ..... 661, 662
size_ub/2 ..... 359, 367
sizes of terms 73
skip 54
skip_code/1 ..... 169, 171
skip_code/2 ..... 169, 171
skip_line/0 ..... 169, 171
skip_line/1 ..... 169, 171
SmallerThan(X, Y) 519
smooth/1 ..... 884, 890
Socket implementation 763
Socket interface 9, 643
socket_accept/2 ..... 473, 474
socket_recv/2 ..... 473, 476
socket_recv_code/3 ..... 473, 475, 476
socket_send/2 ..... 473, 475
socket_shutdown/2 ..... 473, 476
socket_type/1 ..... 473, 477
sockets 473, 1108
sockets/sockets .. 479, 697, 763, 771, 929, 935, 937, 939, 961
sockets/sockets_io 763
sockets_io 479
soft side-effects 366
Solaris 1106
solutions/2 ..... 359, 364
Some scoping issues 527
sort 211, 218, 233, 249, 295, 359, 447, 493, 582, 793, 797, 801, 803, 809, 813
sort/2 ..... 218, 295
sort_dict/2 ..... 13, 813, 814
source directory 1102
source-level debugger 49, 67
source-level debugging 49, 50, 52, 53, 67, 72
sourcename/1 ..... 47, 159, 164, 165
sourcenames/1 47
space/1 ..... 675, 676
spec/1 393
specifications 73, 339
split/4 ..... 517, 519
spy 56
spy-points 49, 51, 72
spy/1 ..... 50, 51, 57, 61
SQL-like database interface 9, 643
sql__attribute/4 745
sql__relation/3 745
sql_goal_tr/2 743
sql_persistent_tr/2 743
sqltype/1 739
sqltypes 739
sqrt/1 157
srandom/1 ..... 807, 808
stabilize_successor/2 945
standalone compiler 21, 1103, 1105
standalone utilities 9, 817
standard total ordering 135
standard_ops/0 ..... 255, 257
start_socket_interface/2 763
start_thread/1 ..... 546, 560
start_threads/0 ..... 763, 765
start_vrmlScene/4 ..... 1023, 1033
static checks 67
static debugging 73
statistics/0 ..... 226, 323
statistics/2 ..... 226, 323
status bar 68
status, this manual 3
steps/2 ..... 359, 367
steps_lb/2 ..... 359, 367
steps_o/2 ..... 359, 367
steps_ub/2 ..... 359, 368
stop_parse/2 ..... 1049, 1061
stop_socket_interface/0 ..... 763, 764
stream/1 ..... 159, 166
stream_alias/1 ..... 159, 166
stream_code/2 ..... 159, 164
stream_property/2 267
stream_to_string/2 ..... 429, 430
stream_to_string/3 ..... 429, 431
streams 359, 397, 429, 457, 669
streams_basic 159
streams_basic:open/3 427
streams_basic:stream/1 .... 929, 930, 931, 932, 933
string/1 ..... 105, 114
string/3 ..... 401, 403
string_to_file/2 ..... 429, 430
string2term/2 ..... 413, 414
stringcommand/1 ..... 346, 350, 352, 353, 354
strings 401, 429, 493, 697, 703, 713, 857
strip_clean/2 ..... 1049, 1059
strip_exposed/2 ..... 1049, 1059
strip_from_list/2 ..... 1049, 1058
strip_from_term/2 ..... 1049, 1059
strip_interface/2 ..... 1049, 1059

strip_restricted/2 ..... 1049, 1059
struct/1 ..... 105, 108
struct_regexp/1 ..... 1073, 1076
style sheets 21, 27, 1105, 1112
sub-shell 67
sub_atom/4 ..... 141, 149
sub_atom/5 ..... 218, 265
sub_times/3 ..... 1083, 1086
sublist/2 ..... 283, 293
subordlist/2 ..... 283, 293
subterm 56
subtract/3 ..... 787, 788
success assertion 342
success/1 ..... 340, 342, 344
success/2 ..... 340, 342
sum_list/2 789
sum_list/3 ..... 789, 790
sum_list_of_lists/2 ..... 789, 790
sum_list_of_lists/3 ..... 789, 790
super class 619
suspend/0 ..... 546, 567
Swedish Institute of Computer Science 10
sybase2sqltype/2 ..... 739, 741
sybase2sqltypes_list/2 ..... 739, 740
sybasetype/1 ..... 739, 740
symbol/1 ..... 884, 890
symbol_option/1 ..... 225, 323, 326
symbol_result/1 ..... 225, 323, 326
symbolic_link/2 ..... 493, 494
symbolic_link/3 ..... 493, 494
symfnames 427, 737
syntax-based coloring 67
Syntax-based highlighting 67
syntax_extensions 193
SYSCALL/1 ..... 515, 516
system .. 23, 28, 65, 211, 219, 220, 221, 222, 223, 224, 277, 297, 301, 359, 388, 417, 419, 427, 457, 483, 487, 493, 500, 501, 502, 503, 504, 505, 506, 669, 685, 697, 703, 725, 752, 759, 768, 935, 939, 989
system libraries 458
system/1 ..... 221, 301, 311, 503
system/2 ..... 221, 301, 311, 503
system_error_report/1 ..... 219, 301, 320, 500
system_extra 483, 493
system_info 205
system_lib/1 ..... 457, 459

T

T. Chikayama 49, 59
t_conj/1 ..... 435, 438
t_disj/1 ..... 435, 439
tab/1 ..... 169, 173
tab/2 ..... 169, 173
table/1 ..... 905, 906
table_widget1 ..... 853, 905
table_widget2 ..... 853, 909
table_widget3 ..... 853, 911
table_widget4 ..... 854, 913
tablewidget1/4 ..... 853, 905
tablewidget1/5 ..... 853, 905
tablewidget2/4 ..... 853, 909, 910
tablewidget2/5 ..... 853, 909
tablewidget3/4 ..... 853, 911
tablewidget3/5 ..... 853, 911
tablewidget4/4 ..... 854, 913, 914
tablewidget4/5 ..... 854, 913
tag_attrib/1 ..... 716, 1096
tar 1102
target/1 487
tau/1 ..... 359, 368
Tcl/tk interface 9, 643
tcl_delete/1 ..... 690, 692, 693
tcl_eval/3 ..... 12, 690, 692
tcl_event/3 ..... 690, 691, 692, 693
tcl_new/1 ..... 690, 692
tclCommand/1 ..... 692, 694
tclInterpreter/1 ..... 692, 693
tcltk 689, 697
tcltk/2 ..... 697, 698
tcltk/tcltk_low_level 692
tcltk_low_level 697
tcltk_raw_code/2 ..... 697, 698, 857
tcsh 21, 87, 1101, 1104
Technical University of Madrid 9
tell/1 ..... 226, 329
telling/1 ..... 226, 329
term/1 105
term_basic 129
term_compare 135
term_size/2 441
term_typing 121
terminates/1 ..... 359, 368
terms 275, 388, 441, 487, 493, 669, 697, 768, 825, 987, 1089

terms_check 105, 359, 445, 768
terms_file_to_vrml/2 ..... 1001, 1002
terms_file_to_vrml_file/2 ..... 1001, 1003
terms_to_vrml/2 ..... 1001, 1003
terms_to_vrml_file/2 ..... 1001, 1003
terms_vars 359, 447, 582, 813
test assertion 342, 379
test/1 ..... 340, 342
test/2 ..... 340, 342
test_format 915
test_type/2 ..... 359, 368
tester 1077
texec assertion 341
texec/1 ..... 340, 341
texec/2 ..... 340, 341
The CLIP Group 31, 33, 41, 89, 91, 141, 177, 229, 231, 239, 259, 267, 269, 271, 277, 283, 295, 299, 331, 337, 381, 383, 399, 405, 429, 441, 445, 447, 451, 507, 509, 643, 673, 689, 723, 731, 747, 767, 773, 789, 791, 831, 843, 1001, 1069, 1099
this_module/1 ..... 205, 207
throw/1 ..... 177, 178
throws/2 ..... 359, 368
tick_option/1 ..... 225, 323, 327
tick_result/1 ..... 225, 323, 325
time stamp 75
time/1 ..... 224, 301, 302, 506
time_analyzer 1083
time_option/1 ..... 225, 323, 327
time_result/1 ..... 225, 323, 326
times(N) 379
title/1 ..... 865, 869
tk_event_loop/1 ..... 691, 692, 694
tk_main_loop/1 ..... 691, 692, 694
tk_new/2 ..... 691, 692, 694
tk_next_event/2 ..... 691, 692, 695
to_list/2 ..... 919, 920
token_read/3 1065
tokeniser 1065
tokeniser/2 1065
tokenize 245
told/0 ..... 226, 329
Tom Howland 511
top-level 49
top-level shell, starting, unix 22
top-level shell, starting, windows 27
top/2 ..... 919, 922
topd/0 685
toplevel 41, 207
toplevel command args, setting 78
toplevel command, setting 78
toplevel/toplevel 43
touch/1 ..... 493, 499
tour, of the manual 7
trace 50
trace/0 ..... 50, 51, 61
tracertc/0 61
tracing the source code 67
transactional update 723
transient state 725
translation_predname/1 195
transpose/2 ..... 12, 395, 396, 797, 799
tree/1 ..... 1035, 1036
triple/1 795
troubleshooting 1101, 1108, 1111
true assertion 347
true/0 ..... 91, 99, 101, 327
true/1 ..... 340, 347
truncate/1 157
trust assertion 346
trust/1 ..... 340, 346
try_finally/3 ..... 493, 497
try_sols(N) 379
ttr/3 ..... 661, 663
ttydisplay/1 ..... 227, 333, 334
ttydisplay_string/1 ..... 227, 333, 334
ttydisplayq/1 ..... 227, 333, 334
ttyflush/0 ..... 227, 333, 334
ttyget/1 ..... 227, 333
ttyget1/1 ..... 227, 333
ttynl/0 ..... 227, 333
ttyout 59, 211, 227, 333
ttyput/1 ..... 227, 333
ttyskip/1 ..... 227, 333
ttyskipeol/0 ..... 227, 333, 334
ttytab/1 ..... 227, 333
type 8, 337
type declarations 821
type of version control 76
type/2 ..... 121, 127
type_compatible/2 ..... 739, 740
type_union/3 ..... 739, 740
types 73

U

U. of Arizona 10
ugraph/1 ..... 687, 797, 799
ugraph2term/2 ..... 685, 686
ugraphs 797, 803
umask/2 ..... 222, 301, 308, 504
uncycle_term/2 449
undo/1 577
undo_force_lazy/1 ..... 43, 46
unfold_tree/2 ..... 1089, 1090
unfold_tree_dic/3 ..... 1089, 1091
uni_type 681
uni_type/2 ..... 675, 681
unify 56
unify_with_occurs_check/2 ..... 217, 265, 266
uninstalling 1102, 1105
union/3 ..... 283, 293
union_idlists/3 ..... 787, 788
unit tests 379
unittest 379
unload/1 ..... 43, 45, 297, 298
unlock_atom/1 ..... 465, 468
unlock_file/2 433
unmarshalling 8, 381
unregister_module/1 ..... 487, 490
unzip 1111
up_to_date 492
up_to_date/2 492
update/0 ..... 457, 458, 459
update_assoc/5 ..... 777, 783
update_attribute/2 201
update_files 725
update_files/0 ..... 725, 728
update_files/1 ..... 725, 728
updated state 724
Updates to persistent predicates 723
UPM 10
url_info/2 ..... 703, 708, 709
url_info_relative/3 ..... 703, 709
url_query/2 ..... 12, 703, 707
url_query_amp/2 ..... 703, 707
url_query_values/2 ..... 12, 703, 707, 708
url_term/1 ..... 715, 720
usage 340
usage relationship 629
use_active_module 585
use_active_module/2 ..... 586, 587
use_class/1 ..... 618, 629, 631, 632, 635, 638
use_compiler/1 664
use_compiler/2 ..... 664, 665
use_foreign_library/1 663
use_foreign_library/2 663
use_foreign_source/1 663
use_foreign_source/2 663
use_linker/1 665
use_linker/2 665
use_module 585
use_module/1 .. 34, 35, 41, 43, 93, 166, 212, 297, 457, 461, 629, 638, 639
use_module/2 ..... 43, 44, 93, 212, 297
use_module/3 ..... 297, 481
use_package 51
use_package/1 ... 43, 45, 97, 211, 239, 511, 725, 733
user module 41, 91
user modules, debugging 49
user setup 21
user:file_alias/2 427
user_output/2 ..... 359, 369
users mailing list 1115
using alternate engines or libraries 37
using_windows/0 ..... 219, 301, 318, 501

V

valid_attributes/2 ..... 915, 917
valid_base/1 ..... 141, 149
valid_format/4 ..... 915, 916
valid_table/2 ..... 915, 917
valid_vectors/4 ..... 915, 917
value_dict/1 ..... 715, 720
var/1 ..... 121, 351
variable instantiation 73
variable names 339
variables 55
variant/2 445
varnamedict/1 ..... 813, 815
varnames/dict_types 813
varnamesl2dict/2 ..... 813, 815
vars_names_dict/3 ..... 813, 815
varsbag/3 447
varset/2 447
varset_in_args/2 447
vector/1 ..... 884, 889
vectors_format/4 ..... 915, 916
verbose_message/1 ..... 487, 488

verbose_message/2 ..... 487, 488
verify_attribute/2 ..... 201, 202, 203
Veroniek Dumortier 10
version control 67, 74
version maintenance mode for packages 75
version number 74
version numbering 75
vertices/2 ..... 797, 798
vertices_edges_to_lgraph/3 803
vertices_edges_to_ugraph/3 797
vertices_edges_to_wgraph/3 801
virtual 620
virtual/1 ..... 617, 620
virtual_method_spec/1 ..... 635, 639
vmember/2 ..... 675, 681
vndict 451, 455, 813
vpath/1 ..... 483, 487, 490
vpath_mode/3 ..... 487, 490
vrml_file_to_terms/2 1001
vrml_file_to_terms_file/2 ..... 1001, 1002
vrml_http_access/2 ..... 1001, 1004
vrml_in_out/2 ..... 1001, 1004
vrml_to_terms/2 ..... 1001, 1003
vrml_web_to_terms/2 1001
vrml_web_to_terms_file/2 ..... 1001, 1002

W

wait/3 ..... 221, 301, 313, 502
waiting/1 ..... 546, 566
wakeup_exp/1 582
WAM 10
warning/1 ..... 197, 198
warning_message/1 ..... 405, 406
warning_message/2 ..... 405, 406
warning_message/3 ..... 405, 406
Web interface 9, 643
WebDB 724
weekday/1 ..... 715, 721
wellformed_body/3 ..... 213, 239, 244
wgraphs 801
when 581
when/2 ..... 581, 582
where/1 ..... 457, 458
whitespace/1 ..... 1035, 1036
whitespace/2 ..... 401, 402
whitespace0/2 ..... 401, 402
why the name Ciao 5
Win32 35
windows shortcut 1111
winpath/2 ..... 219, 301, 318, 501
winpath/3 ..... 219, 301, 319, 501
winpath_c/3 ..... 219, 301, 319, 501
WinZip 1111
Wlodek Drabent 10
word-help.el 68, 70
working_directory/2 ..... 222, 301, 309, 503
write 55, 197, 198, 211, 215, 216, 231, 249, 277, 383, 405, 451, 493, 675, 685, 692, 697, 752, 989, 1005, 1007, 1009, 1013, 1021, 1039, 1041, 1047, 1049, 1077, 1083
write/1 ..... 56, 197, 198, 216, 249, 251, 254
write/2 ..... 216, 249, 250, 473, 474
write_assertion/6 455
write_assertion/7 455
write_assertion_as_comment/6 455
write_assertion_as_comment/7 ..... 455, 456
write_assertion_as_double_comment/6 ... 455, 456
write_assertion_as_double_comment/7 ... 455, 456
write_attribute/1 ..... 215, 249, 253
write_c/write_c 669
write_canonical/1 ..... 215, 249, 252
write_canonical/2 ..... 215, 249, 251
write_list1/1 ..... 215, 249, 252
write_option/1 ..... 216, 249, 250
write_pr/2 979
write_string/1 ..... 401, 402
write_string/2 ..... 401, 402
write_term/2 ..... 216, 249
write_term/3 ..... 216, 249
write_terms_file/2 ..... 1039, 1040
write_vrml_file/2 ..... 1039, 1040
writef/2 ..... 493, 498
writef/3 ..... 493, 498
writef_list/2 ..... 493, 499
writef_list/3 ..... 493, 498
writeq/1 ..... 197, 198, 216, 249, 251
writeq/2 ..... 216, 249, 251
WWW 1103
WWW browser 21, 1105
WWW, interfacing with 701

X

xbarelement1 ... 865
xbarelement1/1 ... 865, 869
xbarelement2/1 ... 871, 873
xbarelement3/1 ... 877
xbarelement4/1 ... 881
xdr_handle ... 1089
xdr_handle/xdr_types ... 1089
xdr_node/1 ... 1089, 1090
xdr_tree/1 ... 1089
xdr_tree/3 ... 1089
xdr_xpath/2 ... 1089, 1091
xdr2html/2 ... 1089, 1090
xdr2html/4 ... 1089, 1090
xelement/1 ... 904
xemacs ... 87
XML ... 701, 703
xml_index/1 ... 1093, 1095
xml_index_query/3 ... 1093, 1095
xml_index_to_file/2 ... 1093, 1095
xml_parse/3 ... 1093, 1094
xml_parse_match/3 ... 1093, 1094
xml_path ... 1093
xml_path/xml_path_types ... 1093
xml_query/3 ... 1093, 1096
xml_search/3 ... 1093, 1094
xml_search_match/3 ... 1093, 1095
xml2terms/2 ... 703, 704
xrefs/xrefsread ... 825

Y

yelement/1 ... 865, 867
yes ... 377, 378